On Timing in Technical Safety Requirements for Mixed-Critical Designs

Von der Fakultät für Elektrotechnik, Informationstechnik, Physik
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines Doktors

der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von Mischa Dominik Lukas Friedrich Möstl

aus Neustadt a.d. Waldnaab, Deutschland, Europäische Union, Erde, Milchstraße

Eingereicht am: 28.06.2021

Mündliche Prüfung am: 04.08.2021

1. Referent: Prof. Dr.-Ing. Rolf Ernst

2. Referent: Prof. Marco Di Natale

Druckjahr: 2021

Acknowledgements

This thesis is the result of eight awesome years at IDA. Such a long time naturally has its bright and shiny days but also dark and cold ones. People come, bring happiness and stay along for the ride. Yet, on rare occasions people also choose to leave. This here, however, is about the bright and shiny days and the people that make it possible to turn the dark and cold times into bright and shiny ones – even if it was not for the entire way. There is a lot that I am deeply grateful for and to which a lot of people contributed. Probably so many that naming them all would greatly exceed this page. So even if you are not explicitly named here, but shared at least an honest smile or laugh with me in these years, please accept my sincerest gratitude. Needless to say, that I remember a lot of these moments!

My scientific journey did not start at IDA eight years ago. In fact, it did start when I was a little boy. My family always encouraged and supported my thirst to explore the world while I grew up to being the person that I am now. They did so, even when my "career" took some setbacks. I partially owe the luck that I eventually became a (researching) engineer rather than an explorer to their endless support. But this big *thank you* goes to more people than just my parents. It goes to the ones I hold dearest and without whom my life would not be as joyful: my family and close friends! Thank you for caring for me, supporting me, and believing in me, especially in times when I could not – thank you for your unconditional love.

But back to the endeavor that resulted in this thesis: If it would not have been for Prof. Ernst, I probably would have never made my way to Braunschweig, a place where I found love and friendship. The first part of this *thank you* to him is for being one of the fathers of the "Informations-Systemtechnik" program , which brought me to Braunschweig. The second part is for fostering me as a student already in the Bachelor program, eventually bringing me aboard the IDA-Team, and being a motivator and "door opener" in all these years. My gratitude in this respect extends to the whole "true" IDA-Team of both floors, including former and current members of scientific staff, shops and administrative staff. You created a work-time family feeling. A feeling that probably not many people share w.r.t. their work place. It would not have been such a joyful time without all of you. Special thanks go to Johannes for his friendship and sharing an office with me for all these years, discussing every idea I voiced in this office with a sharp and open mind. Together with Johannes, I had the honor of spending six years in the research unit CCC. This would have not been possible without Prof. Marco Di Natale,

who not only reviewed CCC but also agreed to the burden of reviewing this thesis. The scientific part of CCC alone was a great time, but what made it an awesome experience was working with Marcus, Alexander, Inga, Mark, Anna and Sophie that helped carve and shape the ideas that are at the foundation of this thesis. To make it plain and simple: *Thank you!*

Abstract

Traditionally, timing requirements as (technical) safety requirements have been avoided through clever functional designs. New vehicle automation concepts and other applications, however, make this harder or even impossible and challenge design automation for cyber-physical systems to provide a solution. This thesis takes upon this challenge by introducing cross-layer dependency analysis to relate timing dependencies in the bounded execution time (BET) model to the functional model of the artifact. In doing so, the analysis is able to reveal where timing dependencies may violate freedom from interference requirements on the functional layer and other intermediate model layers. For design automation this leaves the challenge how such dependencies are avoided or at least be bounded such that the design is feasible: The results are synthesis strategies for implementation requirements and a system-level placement strategy for run-time measures to avoid potentially catastrophic consequences of timing dependencies which are not eliminated from the design. Their applicability is shown in experiments and case studies.

However, all the proposed run-time measures as well as very strict implementation requirements become ever more expensive in terms of design effort for contemporary embedded systems, due to the system's complexity. Hence, the second part of this thesis reflects on the design aspect rather than the analysis aspect of embedded systems and proposes a timing predictable design paradigm based on System-Level Logical Execution Time (SL-LET). Leveraging a timing-design model in SL-LET the proposed methods from the first part can now be applied to improve the quality of a design – timing error handling can now be separated from the run-time methods and from the implementation requirements intended to guarantee them. The thesis therefore introduces timing diversity as a timing-predictable execution theme that handles timing errors without having to deal with them in the implemented application. An automotive 3D-perception case study demonstrates the applicability of timing diversity to ensure predictable end-to-end timing while masking certain types of timing errors.

Zusammenfassung

Traditionell wurden Timing-Anforderungen als (technische) Sicherheitsanforderungen durch geschickte funktionale Entwürfe vermieden. Neue Fahrzeugautomatisierungskonzepte und Anwendungen machen dies jedoch schwieriger oder gar unmöglich; Aufgrund der Problemkomplexität erfordert dies eine Entwurfsautomatisierung für cyber-physische Systeme heraus. Diese Arbeit nimmt sich dieser Herausforderung an, indem sie eine schichtenübergreifende Abhängigkeitsanalyse einführt, um zeitliche Abhängigkeiten im Modell der beschränkten Ausführungszeit (BET) mit dem funktionalen Modell des Artefakts in Beziehung zu setzen. Auf diese Weise ist die Analyse in der Lage, aufzuzeigen, wo Timing-Abhängigkeiten die Anforderungen an die Störungsfreiheit auf der funktionalen Schicht und anderen dazwischenliegenden Modellschichten verletzen können. Für die Entwurfsautomatisierung ergibt sich daraus die Herausforderung, wie solche Abhängigkeiten vermieden oder zumindest so eingegrenzt werden können, dass der Entwurf machbar ist: Das Ergebnis sind Synthesestrategien für Implementierungsanforderungen und eine Platzierungsstrategie auf Systemebene für Laufzeitmaßnahmen zur Vermeidung potentiell katastrophaler Folgen von Timing-Abhängigkeiten, die nicht aus dem Entwurf eliminiert werden. Ihre Anwendbarkeit wird in Experimenten und Fallstudien gezeigt.

Allerdings werden alle vorgeschlagenen Laufzeitmaßnahmen sowie sehr strenge Implementierungsanforderungen für moderne eingebettete Systeme aufgrund der Komplexität des Systems immer teurer im Entwurfsaufwand. Daher befasst sich der zweite Teil dieser Arbeit eher mit dem Entwurfsaspekt als mit dem Analyseaspekt von eingebetteten Systemen und schlägt ein Entwurfsparadigma für vorhersagbares Timing vor, das auf der System-Level Logical Execution Time (SL-LET) basiert. Basierend auf einem Timing-Entwurfsmodell in SL-LET können die vorgeschlagenen Methoden aus dem ersten Teil nun angewandt werden, um die Qualität eines Entwurfs zu verbessern – die Behandlung von Timing-Fehlern kann nun von den Laufzeitmethoden und von den Implementierungsanforderungen, die diese garantieren sollen, getrennt werden. In dieser Arbeit wird daher Timing Diversity als ein Thema der Timing-Vorhersage in der Ausführung eingeführt, das Timing-Fehler behandelt, ohne dass sie in der implementierten Anwendung behandelt werden müssen. Anhand einer Fallstudie aus dem Automobilbereich (3D-Umfeldwahrnehmung) wird die Anwendbarkeit von Timing-Diversität demonstriert, um ein vorhersagbares Ende-zu-Ende-Timing zu gewährleisten und gleichzeitig in der Lage zu sein, bestimmte Arten von Timing-Fehlern zu maskieren.

Contents

Chapter 1

Introduction – New Challenges for Design and Operation

1.1 The Artifact

So what is the artifact in times of the Internet of Things (IoT) and the dawn of connected automated vehicles? For an engineer the most generic explanation probably is, the artifact is *the thing being built*. Consequently, every *thing* is first described by a model. In a very broad sense this claim is universally true, even if you consider yourself a hands-on person that rather crafts and tinkers. There is *always* the thought of what you build – not to every last detail – but the initial thought is there. It is part of human ingenuity, according to Lee in [Lee17]. From a scientific point of view, Lee and Sirjani [LS18] refer to these models as *engineering models*. They specify the design intent and hence are a **specification**. But the *thing* has another angle to it, and that is what has actually been built. Models that try to describe this are referred to by Lee as *scientific models*. "What has actually been built" is also the internal and external behavior of the thing. Hence, scientific models try to analyse an existing thing and describe and understand its behaviors in every detail, while the engineering model(s) have the purpose of conveying a specific intent. Hence, multiple engineering models for the same *thing* exist. Consider a microchip: While the engineer responsible for the package

only has to know which taps from the wafer have to be connected to which pin, she or he does not need to know the exact location of every transistor on the chip; in fact this would even be counterproductive. However, the engineers which design the lithography masks do require this information. Hence, engineering models are often layered as illustrated by Fig. 1.1, where a less detailed model A serves as a specification for another more detailed mo

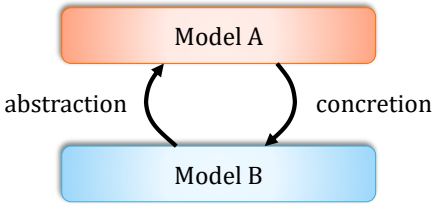


Figure 1.1: Example of layered engineering models: Model A is an abstraction of Model B

For the field of computing, electronic design automation (EDA) emerged and brought forward model verification tools to ensure soundness of model abstractions and concretion. While these EDA tools reduce the factor of human error significantly they are unable to ensure certain properties or the absence of a certain property. One such property is safety. Generically speaking, safety is the property of the absence of harm. Harm can be caused by certain behaviors of a system, consequently it is necessary to have an understanding of the behavior of a system. This results in the need of the scientific modelling of the artifact as engineering models can only specify the intended behavior, but not unintended behaviors.

An inevitable (side) effect in system design is that the hardware architecture and software architecture chosen during the engineering efforts define the possible behavior space of the resulting system. Fig. 1.2 shows an abstract exemplary behavior space, that w.l.g. represents the space of possible behaviors as a two-dimensional space. In this possible behavior space typically only a very narrow area resembles the design goal. This is the area which is described by the specification. However, the resulting behavior space of an artifact can not be simply *selected*. It is rather the result of the design and implementation process. The resulting challenge for the design and implementation efforts is to build the system such that it only exhibits behavior desired by the specification. We acknowledge here that there are two angles to the problem:

1. Coming up with an unambiguous and correct (function) specifications as well as implementing it.
2. Integrating hardware and software into one artifact that matches the requirements in the specification.

The first one is predominantly an issue of languages with unambiguous semantics and rigorous design efforts for individual components – combined with the challenge of translating often natural language requirements and human intent into these languages. The second one, the integration challenge, has to deal with the issue that the composition of sub-systems does not lead to violations in the properties of individual components and hence of the system as a whole. To prevent the latter, designers have to ensure that the integration does not lead to interference between subsystems due to unintended or overlooked dependencies.

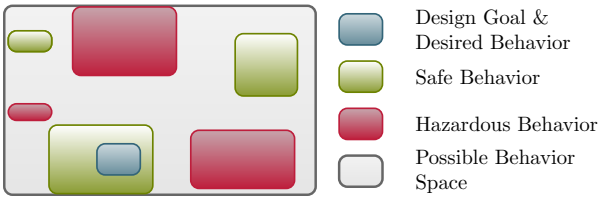


Figure 1.2: Possible behavior space of a design artifact with diffrent regions.

It is evident to anybody who has ever implemented a system, that the borders of any region in Fig. 1.2 are fuzzy. Reasons for this are, that specifications in the first place might be imprecise as they are often in natural language and not in a more formal language. Another reason, is that the consequences of design decisions on the resulting system behavior are not always obvious. A particular instance of this challenge is the integration of functionally unrelated software on a common execution platform. Here, side effects from the execution of one component can lead to unintended behavior in the other. Note that this is not restricted to functional requirements but particularly extends to extra-functional requirements as well. This is the case especially for complex products that are comprised of many sub-systems. At the heart of design methods is the goal to know the area boundary of behavior regions, like in Fig. 1.2, better and sharper. Efforts in the fields of *rigorous design* and design with the aid of *formal verification* as proposed by [Sif18] certainly improve the situation but are to this day not the “silver bullet”.

As a striking example, consider embedded systems in aviation. Besides the ARINC 653 standard [ARI19] for resource sharing in time and space which can be resource inefficient, the trend towards more rigour in the design has left the aviation industry in a difficult position. Certification bodies practically banned computing architectures that are able to deliver the required computing power for future applications, due to their unpredictability [Cer14].

Similarly, the automotive industry currently experiences a huge demand in computation power for future SAE level 3+ vehicles [Int18a]. In contrast to the aviation industry, however, rigorous certification by certification bodies is not required – instead the ECE homologation process is in place. While at the same time less rigour in the design is prevalent, complexity of the runtime software paramounts with the introduction of the AUTOSAR Adaptive Platform and heterogeneous multi-core system-on-chips (SoCs). These boundary conditions lead to a situation where the issue of timing predictability and the necessity of predictable timing for safety of the system are either neglected or deliberately overlooked.

1.1.1 The Necessity of Safety

One of many products that has been in a constant revolution since the introduction of computing are cars. While cars traditionally have been a mechanical product, first electronics where introduced in the late 1960s for injection control [Sim]. Soon more complex electronically controlled functions where introduced, such as anti-lock breaking in the early 1970s [Kuh]. Like anti-lock breaking more and more feature-rich and complex but on the other hand also safety-critical functions in the form of driver assistance systems have emerged, e.g. traction control and its descendant electronic stability control which is nowadays mandatory for new cars in the Single European Market. It is envisioned that the trend towards more and more automation will fully take the driving task away from the human driver with SAE level 5 vehicles [Int18a]. With the introduction of Advanced Driver Assistance Systems (ADAS) and the constant strive for more autonomy on higher SAE levels the challenge to design *safe* systems is present. W.r.t. understanding the behavior of the system this creates a challenge on a much larger scale. It not only requires that non-functional properties of software, like execution time have to be understood, but also functional properties for tasks like environment perception, object detection and classification. Here

the scope is limited to the extra-functional properties and their influence on the safety of the system.

Commonly, safety is defined as the absence of *unreasonable* risks, whereas unreasonable risks are shall be interpreted in a *"certain context and according to valid societal moral concepts"* [Int18b, clause 3.175]. Risk is a combination of the probability of the occurrence of harm to people or the environment, and the severity of that harm [Int18b, clause 3.127]. To better cope with systems that involve multiple risks of different likelihoods, the common safety standards discretize risk. This is for instance done in the general IEC 61508 [The10, part 3] from which ISO 26262 for road vehicles below 3.5 t is a derivative [Int18b, part 3]. Depending on the operational mode of a system, either demand-based or in continuous operation, the probability of occurrence is measured in probability of failure per hour (PFH) or the probability of failure on demand. As assessment of small likelihoods is generally difficult for human designers, the probabilities are discretized into individual safety levels. ISO 26262 does not provide normative nor informative mapping of Automotive Safety Integrity Level (ASIL) to Safety Integrity Level (SIL), but rather generally defines risk as the product of *expected loss in case of accident* times the *probability of the accident occurring* – this is similar to IEC 61508 [The10]. This can be also read as:

$$Risk = Severity \times (Exposure \times Likelihood) \tag{1.1}$$

As already mentioned, small likelihoods are hard to quantify and grasp for humans. Hence, ISO 26262 expresses them in discrete classes of *Controllability* for an average human driver. Hence, it yields:

$$Risk = Severity \times (Exposure \times Controllability) \tag{1.2}$$

Based on the resulting risk, the necessary level of risk reduction can be determined to meet the socially accepted risk level for the design to be seen as "safe". To which rigour the risk reduction has to be conducted, is hence captured in the ASIL. The standard in Part 3 provides a table (Table 4) to compute this based on severity, exposure and controllability classes. It is depicted in Fig. 1.3. The process determines the ASIL of a top-level safety requirement (i.e. the safety goal [Int18b, Part 1, Clause 3.138]) as a result of the hazard analysis and risk assessment (HARA) which reduces the risk to

a socially accepted level – the necessary risk reduction is described by the ASIL. Fig. 1.4 illustrates this flow.

Severity class	Exposure class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A[a]
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 1.3: ASILs according to Part 3 of ISO 26262 according to Eq. (1.2)

The fundamental concept behind safety standards like ISO 26262 in the automotive domain or IEC 61508 for any kind of electrical/electronic/programmable electronic safety-related system for which no particular domain specific standard applies is that any safety-related system must work correctly or fail in a predictable (safe) way. Correctness of a system in this context is the adherence to a specification. Establishing this correctness, however requires that a system's behavior can be described predictably. In that sense, predictability of all possible behaviors becomes a necessity of design for safety-related systems. Avoiding *uncertainty* of behavior either in fault free cases (nominal operation) as well as under fault situations becomes one of the greatest design challenges. Since also the function behavior domain becomes more and more complicated along with the specification the of potential functional failures, initiatives are under way to limit the safety scope to *Safety of the Intended Functionality (SOTIF)*. This initiative has already resulted in a publicly available specification [Int19], however, it has not matured into normative standardization yet.

1.1.2 Example

In the following an example for a state-of-the-art ADAS system from an automated research vehicle will illustrate the already mentioned problems [Ber15]. Fig. 1.5 show the functional architecture of the lateral guidance

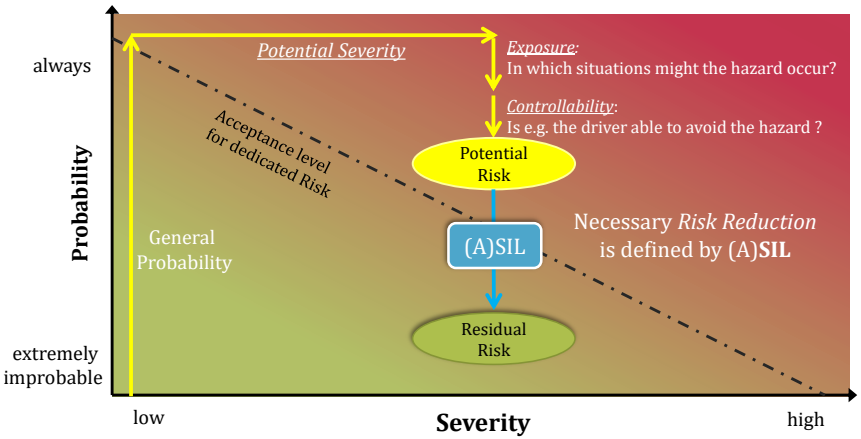


Figure 1.4: The yellow path illustrates how the HARA process determines a potential risk: The ASIL assigned on the safety goal [Int18b, Part 1, clause 3.138] which is the result of the HARA, defines the necessary rigour with which functional safety requirements need to reduce the risk such that a socially accepted risk level is achieved (Figure inspired by [FAH14])

function, while Fig. 1.6 shows a possible implementation in three tasks τ_2, τ_3, τ_5 on two separate electronic control units (ECUs).

An inertial measurement unit (IMU) provides motion data, which is fed to a proper-motion estimation and control algorithm. The control algorithm computes the steering angles, which are used for the control of the actuation. The software driver collecting the data from the IMU is implemented in τ_2. Task τ_3 samples the IMU data (indicated by the dashed line) and performs proper-motion estimation, while the control of actuation is implemented in τ_5. For this example, we assume that data passing from τ_3 to τ_5 also activates τ_5 (solid line).

Typical safety-relevant timing properties are the response time of individual tasks, end-to-end latencies, and the maximum age of data communicated along a task chain. Such bounds can be determined by functional testing of the control algorithm, e.g. with SIMULINK models containing delay and hold elements. A HARA process for our example has determined a maximum tolerable overshoot over the reference value of 0.1 m [SBM16]. In the given implementation the maximum data age between τ_2 and τ_3 and the end-to-end latency from the activation of τ_3 to the termination of a subsequent job of τ_4 must not exceed given bounds to prevent this. This can be shown

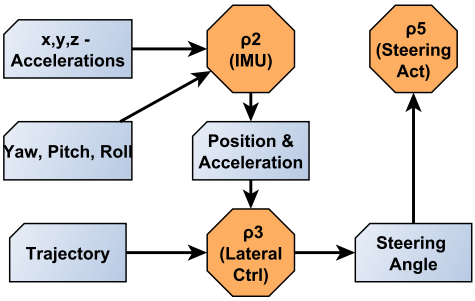


Figure 1.5: Schematic sketch of a lateral control function, realized by three software blocks ρ_2, ρ_3 and ρ_5. The data labels they exchange are depicted in the blue boxes.

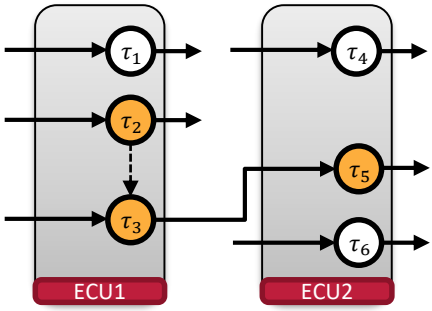


Figure 1.6: Example system with two resources scheduled under static-priority preemptive scheduling, with priorities descending from τ_1 to τ_6. τ_5 is activated by the termination of τ_3. The dashed line indicates sampling of data by τ_3 from τ_2. The orange tasks τ_i implement the software blocks ρ_i from the previous figure.

by functional testing of the lateral controller in SIMULINK. Fig. 1.7 shows two scenarios: The blue line shows the behavior of the controller, if its implementation matches the execution behavior expected by SIMULINK. The pink dashed line, however shows the behavior of the controller if the input data from the IMU is a sample cycle too old. This causes an overshoot over the margin of 0.1 m as specified by the functional safety goal.

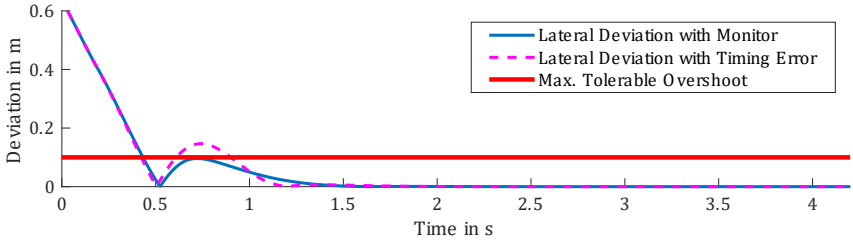


Figure 1.7: Possible lateral deviation under timing errors of the vehicle and maximum tolerable overshoot according to [SBM16]

The data age, also known as freshness, is the maximum distance between a read event of τ_3 and the adjacent preceding write event of τ_2. Hence, the timing behavior of the implementation is not a *non-functional* property but an *extra-functional* property. This extra-functional property is characterized by the fact that violations can cause an alteration in the function. In the concrete example here, the control will not respond as expected by its designer due to the way it is implemented in the shared platform.

As the target platform is shared with tasks (τ_1, τ_4 and τ_6) that implement other functions, these tasks can influence the timing behavior of the tasks implementing the lateral control chain. Let us assume that the task τ_1 and τ_4 implement functions without safety requirements and τ_6 a safety-relevant but less critical function than the chain under consideration. Due to the assumed static-priority preemptive (SPP) scheduling in the example [AUT19b] , both timing requirements transitively depend on the behavior of τ_1 and τ_4 respectively. E.g. τ_1's executions preempt τ_2 and τ_3, and consequently influence the response times of τ_3 and the times when τ_2 writes data and τ_3 reads data. Consequently, the safety of the lateral guidance function implemented by τ_2, τ_3 and τ_5 is not only dependent on their own (timing) behavior but also on τ_1's and τ_4's.

For much simpler lane keeping assistant (LKA) systems of SAE level 2-3 vehicles, lateral control is already rated as an ASIL-D function [BYRLB18], due to the expectable misuse by the driver, e.g. through lacking oversight

over the technical system. Hence, we can conclude that in a conditionally or fully automated vehicle the functional safety requirement will be similar, due to the absence of the human driver. However, there exists a huge difference w.r.t. failure handling. While in the concept for non-automated vehicles [BYRLB18][WHLS16] the human driver can intervene, this is no option in automation scenarios. As a consequence the timing behavior also becomes safety critical, as no safety mechanism like the human driver is present that can turn the behavior into a safe-failure. Due to this impact and the paradigm change it entails in the design of these systems, we revisit this example in the remainder of this thesis where appropriate. This challenge is also fuelled by the fact that the computing hardware (e.g. performance-platforms with dedicated accelerators) as well as the software middleware (e.g. AUTOSAR AP, hypervisors/virtual machines) become ever more complex.

1.2 The Process

Functional safety is not a property that a system, i.e. the artifact, either has or not. It is rather an emergent property that has its roots already in the architecture of a design and the formulation of the requirements for it. This fundamental thoughts are also reflected in the established safety standards like ISO 26262 or IEC 61508 [Int18b] [The10]. In both a significant proportion is dedicated to how the design (incl. requirements formulation and tracing) and implementation *process* has to be structured and how this process shall prevent design faults. The underlying general process is the *V-model*. We shall see that this development model in its original form is becoming less and less suitable for upcoming performance architectures with limited (timing) predictability and operating systems with dynamic process behavior.

1.2.1 The V-Model and its Process

The general process intended by the V-model depicted in Fig. 1.8 is a top-down development flow. It starts with a concept phase that defines the base parameters that the system under development shall fulfill and yields into a requirements and architecture definition phase at the top of the descending branch of the V. The steps atop the left V-branch are the initial steps of the *design phase*. W.r.t. functional safety, all the functional aspects are clarified and defined here. Note, that functional in the understanding of e.g. ISO

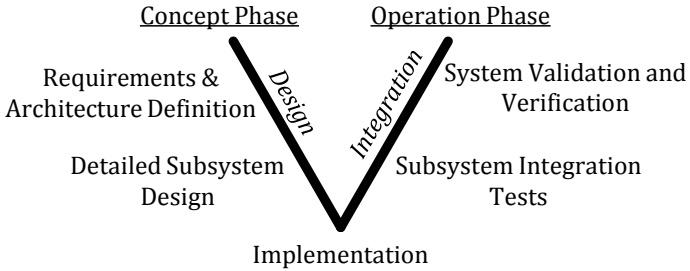


Figure 1.8: V-Model

Checking the designed and implemented system according to this is what happens atop the right branch of the V. If a mismatch occurs the process has to jump back to the opposite side of the V, to refine requirements and/or the architecture definition and go through the V again. The traversal of the V includes the already mentioned detailed subsystem design on the lower left branch of the V, the implementation of hardware and software at the tip of both branches, and subsystem integration tests on the lower right branch. Fig. 1.9 shows how the V-model process looks like if a time dimension is added to it. Without automation costly detours and longer time to market are the consequence. Particularly for timing requirements that – as we have already motivated in the previous section – must find their way into safety concepts.

1.2.2 State of the Art in Functional Safety Verification

Functional safety is a prerequisite to qualify products for the market, as e.g. statutory for the Single European Market [PC10]. To narrow the focus

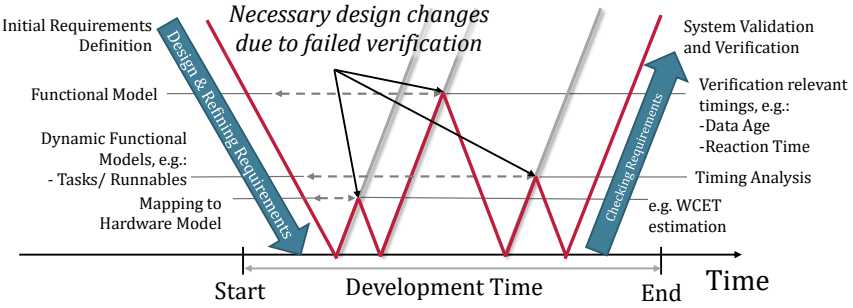


Figure 1.9: What is actually happening in a V-model process: The process is never a V but has a lot of iterations

we concentrate on automotive domain here. Although we have outlined that the aviation domain has processes which seem to require more rigour - they effectively face similar problems. However, the volume and market impact are smaller than for automotive. The state of the art for functional safety processes in the automotive domain is marked by ISO 26262 [Int18b] which provides detailed normative requirements for the process.

The (safety) process begins with a concept phase, that starts even before the V from underlying V-model. As part of this concept phase, a HARA for each function under consideration is performed and the level of required risk reduction is determined based on the yellow flow in Fig. 1.4 that determines the acceptable risk based on societal concepts (cf. Eq. (1.2)). A work product of this process is the amount of necessary risk reduction which is already "measured" with an ASIL. Based on these risks, items that "implement a function or part of the function at the vehicle level, to which ISO 26262 is applied" [Int18b, Part 1, clause 3.82] are defined. Depending on which risks they either carry or mitigate, functional safety requirements for the items are derived and quantified with ASILs. Note that although the item is referred to as an implementation of a function, it is still an abstract, i.e. concrete code and hardware agnostic description – it is just less abstract than a pure function description. How the ASIL for an item is going to be achieved is described in a functional safety concept that details design measures based on the assigned ASILs. From this abstract functional safety concept, concrete *technical safety requirements* need to be derived based on the design of the concrete implementation. These technical safety requirements inherit the ASIL of the functional safety requirement that they implement, if no risk reduction methods can be applied in the design. In the past it typically was

not necessary to specify technical safety requirements on timing, since only a safety function to detect misbehavior was necessary and the function's service could be discontinued, e.g. by handing over control to the driver immediately. However, as we have seen in the example of the lateral controller in Fig. 1.7 such requirements will be necessary for future SAE level 3+ vehicles. Besides specifying the requirements, the process laid out in ISO 26262 specifies quite an extensive number of process requirements, especially if software is subject to ASILs [Int18b, Part 6].

For verifying requirements as well as validating them the design process for hardware, software, and the system level proposes a number of methods to reduce design *uncertainty* and *underspecification*. The two foremost and renown are Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA), whereas the FMEA often comes in the form of a Failure Mode Effects and Criticality Analysis (FMECA). FTA is a top-down, deductive design review method that tries to identify causes of failures originating from a system failure condition [Int06]. The resulting fault tree is thereby typically connected by boolean logic operators, that may allow quantification of the failure event at the top. On the contrary, FMEA is an inductive technique that is used to identify ways a system can fail based on root failure classes of single components [The18]. Both FMEA and FTA are methods that are used for concurrent design review to identify the best ways to reduce risk or to validate a requirement, i.e. check whether it must be replaced with a different one.

However, let us review how the process description from the standard addresses the safety challenge and compare it to how typical engineering projects for safety related systems proceed. ISO 26262 to a large extent acts as if every safety related design is being done from scratch since it is organized strictly top-down. [1] A particular point here is that hardware on which a function shall be implemented is already predetermined at the start of the project, e.g. due to evolution of a legacy product. Furthermore, large parts of the software, like the operating system (OS) and the Run Time Environment (RTE) might also already be fixed, e.g. due to long-term supply contracts, etc. Last but most important, parts of cause-effect chains that are integrated from a legacy product are already fixed and can only be changed under high investment costs. A fact due to which industrial practice refrains

[1] The safety element out of context (SEooC) concept is a clear exception from this – however, in order to certify something as a SEooC a document that describes the boundary conditions under which the SEooC part can be used without recertification is necessary. For complex behavior such as timing this is very hard to achieve as it is dependent on other components of the system (c.f. the other tasks in the lateral controller example from Fig. 1.6).

from changing or redeveloping legacy parts but adapting new developments in such a way that they are compatible to legacy parts. While in the end the normative requirements of standards like ISO 26262 are still fulfilled, it is not necessarily the best engineering solution w.r.t. it.

1.2.3 Current challenges for the Process: Mixed-Criticality and Continuous Integration

The current state of the art of the process reveals a number of shortcomings. First and foremost, it has difficulties handling mixed-critical designs, especially if they are integrated on modern highly-integrated computing architectures. These architectures introduce a range of challenges for which the safety process needs to argue *freedom from interference* [Int18b, Part 1, Clause 3.62]. In designs where only a limited number of functions with the same criticality, i.e. safety impact, are integrated on a single platform, the design can be specifically tailored towards their needs. In complex architectures, isolation mechanisms between the multiple tenants of the platform must be devised in such a fashion that freedom from interference can be guaranteed. If we consider the timing aspect and its safety implications, we can observe that an extensive number of publications in this area are available. [BD19] provides the most recent overview. However, there exists a substantial misinterpretation of the freedom from interference requirement in the mixed-criticality (MC) research community. The common misconception in these works is that a higher level of criticality is allowed to influence a lower level while still operating within the specification. However, as e.g. laid out in [EN16] "absence of cascading failures" [Int18b, Part 1, clause 3.62] is a property that works both ways, i.e. also the lower critical component can expect it from a higher critical one. Hence, based on the ASIL level for timing requirements, ways to limit timing failures appropriately are necessary.

Second, the process suffers from the complexity of the artifact itself. Designs, e.g. automated driving, reach into behavioral regions with higher and higher complexity, which requires more computational performance that comes with higher complexity. This is especially true for software dominated systems. Requirements engineering experts such as [Nan12] conclude that:

> *Almost all accidents with serious consequences in which software was involved can be traced to requirements failures, and particularly to incomplete requirements.*

As we have seen, extensive parts of safety standards deal with requirements validation, i.e. to check whether the right requirements all the way down to the technical level have been formulated. But the challenge of complexity can not be tamed by this approach. Even worse, studies such as [MG17] show that most practitioners tend to use FTA and FMEA in practically basic form. The authors conclude, that only rarely new methods that leverage automation are used, although the systems become software dominated.

Third, the company structure still has a significant impact on how complex systems are structured in their technical realization. This phenomenon has already been observed in the late 1960s by [Con68] but is still partially true today. It can be observed in the architecture of on-board vehicle networks over the last decades. While originally a single bus was sufficient to connect all ECUs and hence functions of a car, where needed, soon multiple buses connected via a gateway for separate departments evolved. Gateways soon became a major engineering challenge and required sophisticated methods to package and handle traffic with ever-increasing amounts of data to exchange [TSAE16a][KRST11]. Intended solutions to this are service-oriented design, where the resource on which a piece of software executes can be arbitrarily determined [WLB+18]. However, this increases the mixed-criticality requirements as a giant bouquet of services has to be handled. The service-oriented design is extended by the idea of a zone-architecture. In this approach a set of high-performance "compute servers" are distributed throughout the car [MBB18]. Yet, it also comes with high mixed-criticality requirements, as all software consolidated on the zone controllers are subject to the freedom from interference constraint mandated by the safety standard.

The fourth challenge is that the complexity of the artifact does not scale in the similar order than the performance of analysis methods to overcome design uncertainty and to detect underspecification. As the prime methods for this are FMEA and FTA the same amount of designers can not handle the increasing complexity well if the methods are not improved for challenges like safety-relevant timing requirements. Consider the FMEA sheet shown in Table 1.1 – which has the same principle elements as mandated by [The18]. The sheet holds a conceptual FMEA for the timing of a cause-effect chain, where the data-age along the cause-effect chain is critical to a safe operation of the function. What is under consideration here, are both – the design and the operation of the cause-effect chain. For the design aspect a worst-case design approach is considered. For the potential failure effect that the function exits the envelope of safe behavior, several potential causes can be determined. These causes all relate to the specification that describes the timing behavior of parts of a system. However, the factors that influence

the timing behavior on the system level are *not composable*, meaning that the timing behavior of one task can not be assessed in isolation but always requires to investigate the composition to reason about its resulting timing behavior. Especially, for end-to-end properties [SE09a][TJJG19]. Hence, this abstract FMEA must be instantiated and performed for each cause-effect chain in a system after any change that has an impact on a specification parameter that has an influence on timing. This procedure gets out of hand very quickly if no automation is in place.

1.2.4 The Research Unit Controlling Concurrent Change

The Research Unit Controlling Concurrent Change (CCC) was funded by Deutsche Forschungsgemeinschaft (DFG) from 2013 until 2019 at TU Braunschweig. In nine subprojects, research was conducted around the idea of an in-field integration process depicted in Fig. 1.10.

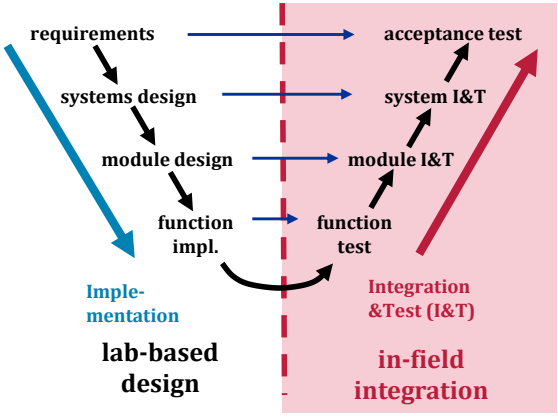


Figure 1.10: Design and integration flow originally envisoned by the CCC project

This process was originally envisioned as a split of the classic V-Model process from Fig. 1.8 at the bottom tip, into a lab-based phase that provides ready to integrate components and an automated in-field phase which integrates these components into a system. The project had two application sub-projects for demonstration and including the perspective of critical systems: space robots [DAF+19][MDA+19] and automated driving [MNSE19]. To perform in-field updates (or changes in general) an architecture has

Table 1.1: Conceptual FMEA for timing of cause-effect chains

Process step	**Potential failure mode**	**Potential failure effects**	**Potential causes**
What is the step?	*In what ways can the step go wrong?*	*What is the impact on the customer if the failure mode is not prevented or corrected?*	*What causes the step to go wrong? How can the failure mode occur?*
Design of the cause-effect chain	Violation of data-age requirement at runtime	Function exiting the envelope of safe operation	Own WCET specification not conservative WCET of scheduling interferer not conservative Own event model specifications not conservative Event model specifications of scheduling interferers not conservative
Operation of the cause-effect chain	Violation of data-age requirement at runtime	Function exiting the envelope of safe operation	Hardware fault causing exceedance of own estimated WCET specification Hardware fault causing exceedance of WCET specification of a scheduling interferer Hardware fault causing non conservative activation pattern for tasks of the cause-effect chain Hardware fault causing non conservative activation pattern for scheduling interferers

been devised that bisects the system into a model domain and an execution domain. It is shown in Fig. 1.11.

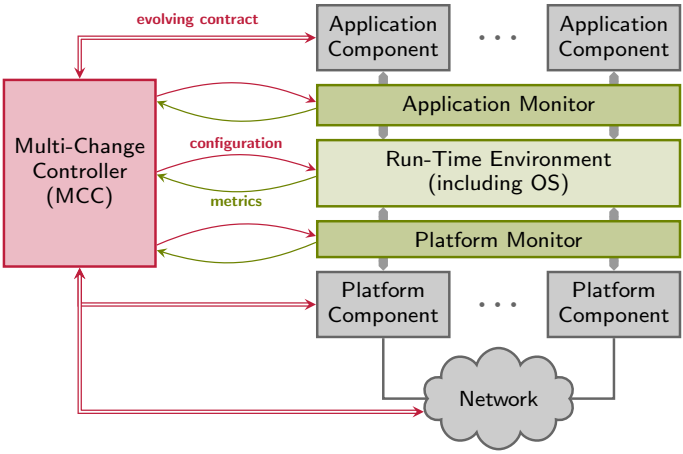


Figure 1.11: CCC system architecture, with the model domain on the left (red) and the execution domain on the right (green). The componets which are subject to change are depicted in gray. [SNM+17]

The model domain, basically consists of an entity referred to as Multi-Change Controller (MCC), which configures the execution domain. The MCC hosts the models describing the system, as well as descriptions of lab-implemented components that can be added to the system. Through analysis of the already existing configuration and descriptions of requested changes the MCC generates new configurations respecting boundary conditions like security, safety and availability, as well as optimizations of the configuration. I was involved with the subproject that dealt with safety concerns in the flow and architecture. While previous work already investigated the use of monitoring and enforcement at run-time, a system-level coordination has not been reported before. CCC changed this. Furthermore, the points already raised in the previous two sections are an initial result of the safety and availability subproject. The work in this subproject culminated in the research questions that follow in the next section and led to the results of this thesis.

1.3 Resulting Research Questions

Based on the challenges for artifact and process in safety related system, the research questions for this thesis can be formulated. The focus is on systems with functions in which the timing of cause-effect chains is safety critical, i.e. generates technical safety requirements as a variation in the timing can lead to an alteration of the function. While race conditions have been a field of interest in computer science for a long time, to the best of knowledge the aspect of timing as technical safety requirements in the context of safety standards has not been thoroughly investigated before. State-of-the-art safety processes follow an argumentation for these functions, where a fail-safe approach is taken since the timing safety issue can not be tamed successfully. Further, the aim here is to enable mixed-criticality workloads on consolidated platforms in such a way that the assurance of the timing of end-to-end cause-effect chains can be raised to their respective safety level. Here, the goal is that either the risk of the resulting failure is sufficiently low or that fail-operational behavior can be achieved.

The basic assumption for functions is that in any case the function is correct and fault free, if the timing is correct, i.e. the approaches presented here do not improve the situation for functional failures. They only tackle the issue of the extra-functional property of their timing, which must be correct to deliver correct service.

Question 1: How can hidden timing dependencies of a design be detected, quantified, and in a multi-layered design model be brought to the attention of the function design team or the team that manages functional safety?

Question 2: How can the design process for mixed-critical systems be supported such that technical safety requirements on timing can be quantified and automatically generated where they are necessary? This specifically targets mixed-critical systems, where only a portion of the cause-effect chains has extra-functional and safety critical timing requirements but is timing wise influenced by other tasks in the system (cf. Fig. 1.6). How must timing safety requirements be allocated in these cases?

Question 3: How can timing engineering decouple dependencies of critical cause-effect chains from other tasks and chains if interferes can not be qualified up to the level that is necessary to argue the independence of the critical chain from the interferes. How can the use of such mechanisms be coordinated at the system level?

Question 1 is addressed by cross-layer timing-dependence analysis which is covered in the next chapter. Questions 2 and 3 are both investigated in Chapter 3, whereas Section 3.1 answers Question 2 and Section 3.2 answers Question 3 based on the system model introduced in Section 2.1.Based on these results, intermediate conclusions are drawn in Chapter 4 and the research questions are extended. As the conclusions in Section 4.1 show that, in the terms of [LS18], the state-of-the art bounded execution time (BET) timing model is a good scientific model but not a suitable engineering model for timing of data, the following research questions are added:

Question 4: How can data-centric effect chains' timing safety requirements be specified, verified and validated?

Question 5: Is there another solution to the problem of needing ever more engineering effort to enable timing-safe fail-operational systems?

The latter two questions are addressed in Chapter 4 and Chapter 5, respectively. Chapter 4 introduces system-level logical execution time (SL-LET) as an engineering model for timing and Chapter 5 proposes *timing diversity* as a mechanism to achieve timing safety based on the engineering knowledge captured in SL-LET models. An overall conclusion covering the findings of this thesis is subsequently presented in Chapter 6.

Chapter 2

Cross-Layer (Timing-)Dependency Analysis

To address the raised research questions, Section 2.1 first introduces the cross-layer model that is used throughout this thesis. The model is later extended in Section 4.3. Section 2.2 presents the basic flow for a dependency analysis to identify the dependencies on other layers of the cross-layer model (CLM) which are abstracted on higher ones. The remainder of the chapter focuses specifically on timing dependencies and shows how to solve the challenge of identifying timing dependencies for an application like the lateral controller from the introduction in Section 2.6. Section 2.4 and Section 2.5 introduce the data structure and analysis methodology. The chapter concludes with an overview of the related work in Section 2.7 which is split into related work on modelling approaches and work related to dependency analyses.

2.1 Cross-Layer System-Model

The literature reports also fairly recent cross-layer modelling approaches to design and develop dependable and embedded systems. Here, a stripped down custom model is chosen for intelligibility and simplicity. The model

described here has a strong focus on timing aspects and timing behavior of systems – a factor often not covered in established models, at least not to the full extent of the state of research in the real-time domain. Nevertheless, the model embraced here follows similar guidelines as design languages like EAST-ADL [Ass13], AADL [AAD17] or EEA-ADL [Mat10] and borrows model elements from them. Related work on these meta-models is presented in Section 2.7.

Basic principles for model development are also proposed in ISO/IEC 42010:2011 *Systems- and software engineering* [Int11]. ISO/IEC 42010 proposes different views on a system by a number of domain-specific models. A *viewpoint* (or aspect) in systems engineering is thereby a convention on how to look at a system. The viewpoint's information can be described in individual models, which are referred to as layers in this thesis. A particular advantage is, that aspects such as functionality, hardware, and software can be conveniently described in expert-grade models. This allows experts to describe design relevant issues and specify architectural viewpoint specific requirements. In [Int11] different architectural viewpoints are connected through correspondence rules between models of individual architectural views. Correspondences allow to describe that model elements from one model, i.e. an architecture viewpoint, can be instances in another model. Correspondence rules describe valid correspondences between individual architecture viewpoints, and hence checking correspondence rules thereby ensures consistency of models describing different architecture views. To simplify the terminology, the terms architecture view and architecture model are used interchangeably.

Although, different architectural models structure systems engineering efforts, they often do not capture an *aspect* – such as timing – in a single model. As an example consider timing of dataflow: While the dataflow itself might be specified in a model-layer, with elements that produce and consume data, the model does not specify the timing behavior of the system. The timing of the execution behavior is specified in a different model. Hence, there are interdependencies between the two models for certain timing aspects, e.g. the latency of data. The requirement is annotated with the model element for the data – checking satisfaction of such a requirement, however, requires to traverse the dependencies between models and putting information from different models into the context of an analysis. This is illustrated in Fig. 2.1 for a data-age requirement. Avoiding design faults in aspects like timing hence requires good navigation helpers in the fog of abstraction created by different models. While the dependency analysis in Section 2.5 is the navigation instrument, the CLM defines the structure and syntax of the map.

In case of the example in Fig. 2.1, the task model and the resource model concertize the timing behavior, however, it is abstracted in the software model.

The scientific goal here is not to create "the best map", i.e. the best model framework where each and every aspect has its dedicated layer. This would only result in consistency challenges between layers. The goal here is, to utilize the structure within and between layers to analyze an aspect with information scattered across layers. In the case of this thesis, this aspect is timing in general. Dependency analysis' intent thereby is to identify which model information might influence the aspect.

Fig. 2.1 sketches the structure of the CLM introduced in the following in more detail. This structure follows the principle of abstraction and concretion as described by Lee et al. [LS18][Lee17] (cf. Fig. 1.1). In the figure, the timing requirement, i.e. the important aspect, is annotated in the software model. Yet the actual timing behavior is described in the task model in conjunction with the resource model. However, as can be seen in the figure, many other inter and intra model dependencies transitively influence the model elements which implement the requirement under investigation. How this issue is systematically addressed is described in Section 2.2.

It shall also be note here, that for Cyber-Physical Systems (CPSs) the CLM is embedded in a model of the physical environment, with which it interacts. This interaction can be perceived, i.e. modeled, in different ways. One approach, that is shown in Fig. 2.1, is to express the coupling through the functional model (i.e. through the functional intent) and the physical model of the system itself, e.g. the sensors and actuators modelled in the hardware model. Physics and the "physical world" dictate (functional) requirements, e.g. the maximum tolerable overshoot of a controller as shown in Fig. 1.7, however, they can also introduce dependencies as discussed in [MSE+18b] and [ME15].

For the purpose of describing systems in this document, the scope of architectural views is limited to four models (cf. Fig. 2.1) that are introduced in the following. Furthermore, we express correspondences (for concretion and abstraction) by so called *mappings* from one element in model A to one or multiple model elements in model B (cf. Fig. 1.1). Each model can in principle be treated as a graph with multiple edge and node types. We illustrate this on the four chosen model layers, which are relevant in the scope of this thesis: a logical function model, software model, task model, and a resource model. While each model internally describes dependencies, the inter-model mappings, e.g. to shared elements, also describe dependencies. Revealing

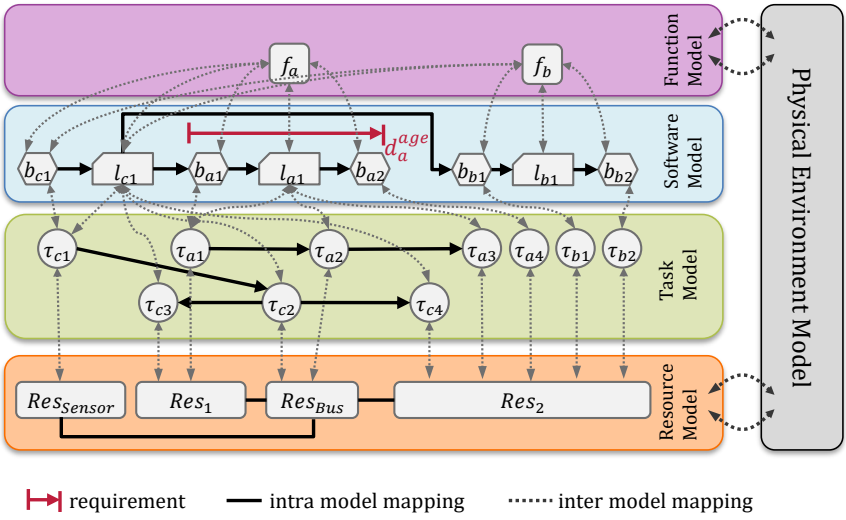


Figure 2.1: Cross-Layer Model Structure where mappings between elements of different layers fulfill the role of abstraction and concretion as described by [LS18]

them and propagating the consequences to the system-level context is part of the dependency analysis described in Section 2.5 and following.

2.1.1 Models

2.1.1.1 Function Model

Particularly in the automotive domain, it is common to structure the design of a vehicle into individual vehicle-level functions. A function thereby is a rather high-level and logical abstraction of functionality at the vehicle level. For instance to carry out the driving task, lateral guidance and longitudinal guidance of the vehicle are necessary. It is obvious that the level of abstraction or detail on the logical function level may vary depending on the use case for the function model. In principle, the main application of the function model is to structure the development process, and to attribute responsibility for implementing them in the resulting system. Formally, we define a function model as follows:

Definition 2.1.1: Function Model

A *function model* is a graph $\mathcal{FG} = (\mathcal{F}, \hookrightarrow)$ where the nodes in $\mathcal{F}$ describe the functions, and $\hookrightarrow$ is the set of edges that describe functional interactions.

The function model hence allows the designer to specify intended functions and their (intended) interactions to achieve the desired behavior at the system level. A particular application of this model is for structuring the development process e.g. the definition of vehicle level functions to enable the development according to safety standards. In the automotive domain, safety goals are defined for individual functions, and functional safety concepts are derived based on the definition of the defined (vehicle-level) functions. It has to be noted that a function is still an implementation agnostic design artifact. Hence, it does not imply any particular implementation or constraint on the implementation yet. The same holds for the functional safety concept [Int18b, part 1, clause 3.65] which specifies functional safety requirements based on safety goals derived in a hazard and risk analysis.

2.1.1.2 Software Model

In the course of the design process, it is decided which functionality is implemented where and how. While some parts of a function are carried out in hardware, others are implemented in software. The latter are described by a *software model*.

Definition 2.1.2: Software Model

A *software model* is a directed graph $\mathcal{SG} = (\mathcal{B} \cup \mathcal{L}, \xrightarrow{w} \cup \xrightarrow{r})$ where the union set $\mathcal{B} \cup \mathcal{L}$ represents the vertices and $\xrightarrow{w} \cup \xrightarrow{r}$ the set of directed edges. $\mathcal{B}$ is the set of software blocks and $\mathcal{L}$ is the set of data labels which semantically identify data exchanged by software blocks. $\xrightarrow{w}$ and $\xrightarrow{r}$ define precedence relations between software blocks and labels and vice versa, denoting writing and reading of labels by software blocks. Each $L \in \mathcal{L}$ can have at most one producer, i.e. there exists at most one $(b_i, L_j) \in \xrightarrow{w} \ \forall L_j \in \mathcal{L}$.

In essence, the software model is a simple data flow model that describes which data labels are produced and consumed by which software block. In

its structure it is similar to the notion of SIMULINK models commonly used in control engineering. The software model does not express any form of timing behavior, however it allows annotating requirements on the timing along a data flow.

Definition 2.1.3: Timing Requirements on the data flow in the software model

(a) the backward distance $d_{i,j}^{age}$ which is defined as the maximum time between a read event of b_j and the preceding write event of b_i on a common label (data age), and
(b) the forward distance $d_{i,j}^{rea}$ which defines the maximum time between a write event to a label by b_i and its subsequent read by b_j (reaction-time).

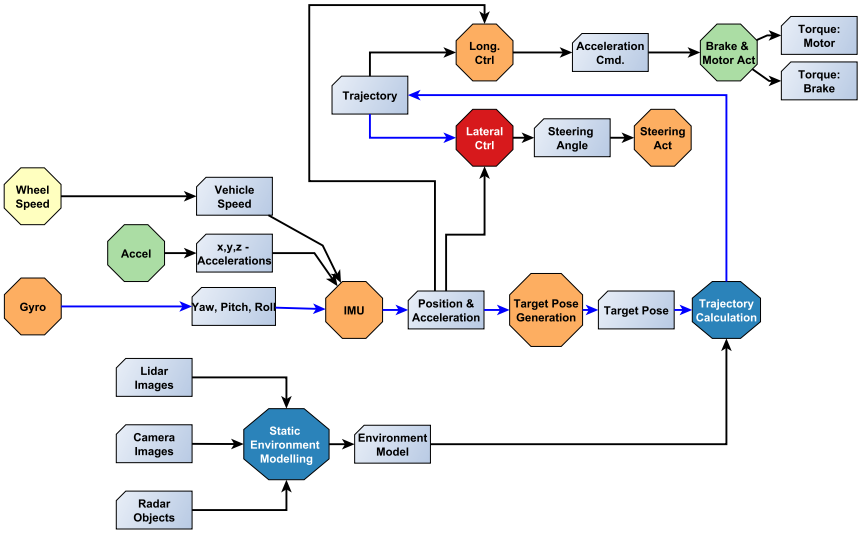


Figure 2.2: Software Model

As an example Fig. 2.2 shows a software model implementing the functional chain from Fig. 1.5 in more detail.

2.1.1.3 Bounded Execution Time Task Model

The timing behavior of a system is described by observable events in time. It is assumed that the workload which a system executes over time can be structured into recurring *tasks*. The concrete instance of a task is referred to as a job, which consumes a certain amount of execution time. Hence, an activation trace can be recorded from the execution of a system, that records how much execution time a job requests.

Definition 2.1.4: Activation Trace

An *activation trace* of a task τ_i is a function

$$\sigma_i : \mathbb{N} \rightarrow \mathbb{R}^+ \times \mathbb{R}^+$$

where $\sigma_i(n) = (t, C)$ denote that the n-th event occurs at time t and requires a processing time C on the resource the task is mapped to.

Like the activation, also the termination of a particular piece of workload, is an observable event in time, captured by the termination trace:

Definition 2.1.5: Termination Trace

An *termination trace* of a task τ_i is a function

$$\omega_i : \mathbb{N} \rightarrow \mathbb{R}^+$$

where $\omega_i(n) = t$ denotes the time when the n-th activation of τ_i is completed.

Note that for an event n with $\sigma_i(n) = (t_0, C)$ and $\omega_i(n) = t_1$ it is not necessarily the case that $t_0 + C = t_1$, as the execution resource might be shared. Hence, for each job $\tau_{i,n}$ the response time can be computed by:

$$R_i = \omega_i(n) - \sigma_i(n) \tag{2.1}$$

These concrete execution traces can be abstracted by the BET model. In this model, the primary assumption is that computational workload is structured such that upper and lower bounds on the needed resources to execute the workload can be provided. As the model assumes that the consumed service from a resource is execution time, lower and upper bounds, referred to as

best-case execution time (BCET) C_i^- and worst-case execution time (WCET) C_i^+, on the execution time in any possible trace σ_i are estimated:

$$C_i^- \models \sigma_i \, , \; C_i^+ \models \sigma_i \tag{2.2}$$

A second assumption is that, tasks release workload according to a pattern that can also be bounded, referred to as an event model. The events in this case are the release of workload, i.e. the activation of a task which spawns a job.

Definition 2.1.6: Event Model

An event model $\langle \delta^+(n), \delta^-(n) \rangle$ is defined by

$$\delta^+ : \mathbb{N}^+ \rightarrow \mathbb{R}_0^+$$
$$\delta^- : \mathbb{N}^+ \rightarrow \mathbb{R}_0^+$$

where δ^+ returns an upper bound and δ^- a lower bound on the time interval between the first and the last event of any sequence of n event arrivals.

Event models as well es execution time bounds are properties of a task τ_i in the BET model. Based on Definition 2.1.4 to Definition 2.1.6 we can formally define a BET task:

Definition 2.1.7: BET-Task

A *task* $\tau_i \in \mathcal{T}$ is defined by a tuple $(C_i^+, C_i^-, \delta_{i,in}^+, \delta_{i,in}^-)$, where C_i^+ specifies the WCET and C_i^- the BCET respectively, $\delta_{i,in}^+, \delta_{i,in}^-$ are event model abstractions of concrete execution traces that capture the maximum/minimum time interval between n consecutive activation events for all n.

The BCET and the WCET describe upper/lower bounds on the execution times C in σ, while the two-tuple of $\langle \delta^+(n), \delta^-(n) \rangle$ constitutes an event model that upper/lower bounds the arrival times in σ. Commonly, it is assumed that both, execution time estimates and event models, are conservative estimates, and hence also the timing model is conservative. In this thesis WCETs / BCETs C_i^- / C_i^+ are not necessarily conservative estimates

in the sense of [WGR+09][Wil18]. Whenever, a timing parameter such as C_i^- or C_i^+ are a conservative estimate, they are explicitly referenced as such.

Definition 2.1.8: BET-Task Job

A concrete instance of a task τ_i, i.e. an activation, is referred to as a job, with $\tau_{i,j}$ denoting the j-th instance of τ_i.

To describe interactions between tasks, i.e. whenever the termination event of one task becomes the activation event of another one, the event dependencies can be captured by a graph structure:

Definition 2.1.9: Task Graph

A *task graph* is a tuple $\mathcal{TG} = (\mathcal{T}, \rightarrow)$ with $\mathcal{T}$ representing the set of tasks, $\rightarrow$ defining a directed precedence relation between tasks such that output/completion events become activation events of the succeding one.

Together with the resource model and the mapping between the two models the task graph forms the timing model to bound timing behavior.

Note that the task graph does not necessarily capture the timing behavior of data flow as there are two fundamentally different forms of cause-effect chains.. Timing behavior of data flow is only described by the model if an event dependency also implies data flow, as it is e.g. the case when CAN buses or Ethernet is modelled. In contrast to that, implicit communication between (event) independent tasks is not captured by the model, e.g. via shared memory as it is commonly done in AUTOSAR classic systems. How particularly cause-effect chains of the latter type are captured in the CLM is detailed in Definition 2.1.12 further on.

2.1.1.4 Resource Model / Physical Model

Further typical models of a cross-layer model are a systematic description of hardware resources in the system and a physical model of the environment the system operates in. For both model types extensive modelling is possible, however, we restrict the scope here to a few properties of the resource model used in the remainder. Note that these restrictions do not limit the generality of the approach.

The resource model embraced here is very simplistic. It consists of a set of resources as nodes and edges that describe connections between the resources.

Definition 2.1.10: Resouce Graph

A *resource graph* is a tuple $\mathcal{RG} = (\hat{\mathcal{R}}, \xrightarrow{Con})$ with $\hat{\mathcal{R}} = \mathcal{R} \cup \mathcal{C}$ representing the sets of processing resources $\mathcal{R}$ and communication resources $\mathcal{C}$, and $\xrightarrow{Con}$ defining the connections/data paths between resources.

Resources can have a number of properties. The first one being that resources provide a certain service, e.g. communication resources like a switch port provide "communication service" while processing resources like a CPU provide processing service. This service is consumed by the tasks mapped to the resources by the resource allocation function ρ. The way service is divided among multiple tasks mapped to a resource is governed by the scheduling policy. Consequently, the scheduling policy is the second important attribute. The third property used in this thesis are possible fault patterns of a resource. This can be expressed, e.g. by an error rate for the resource and whether an error permanently damages a resource and causes it to fail or whether the error is transient and correct state and operation can be restored, e.g. by corrective actions upon detection.

Physical properties of resources and the environment they operate in the scope of a cross-layer model and dependency analysis are e.g. described in [MDA+19]. However, such properties are out of scope for the remainder of this work and are omitted for brevity.

2.1.2 Mappings between Models

In order to form a CLM, the different architectural views expressed by the model graphs must be put into relation. The bidirectional mappings, to express a model correspondence, between two architectural models X and Y follows a semantic. While one direction in the presented CLM is the direction in which elements from X *map to* an element in Y in the sense of *implemented by/on*, the backward direction from Y to X can be interpreted as *hosts* or *provides for*. This semantic of the mapping edges will become crucial for the general dependency analysis flow in Section 2.2. W.r.t. syntactic properties the mapping between a model X and a model Y is bipartite:

Lemma 2.1.11: Bipartite Mapping

A *mapping* from an architecture model X to an architecture model Y is a bipartite graph $M_Y^X = (\mathcal{V}^X, \mathcal{V}^Y, \mathcal{E}_Y^X \cup \mathcal{E}_X^Y)$, where $\mathcal{V}^X$ is the node set of model X and $\mathcal{V}^Y$ is the node set of model Y; $\mathcal{E}_Y^X \cup \mathcal{E}_X^Y$ is the set of bidirectional edges such that $\forall e_{i,j} = (v_i, v_j) \in \mathcal{E}_Y^X \; \exists e_{j,i} = (v_j, v_i) \in \mathcal{E}_X^Y$.

In the following, this is substantiated for the mappings that we assume in the presented CLM.

2.1.2.1 Mapping between Function Model and Software Model

The abstract functions $f \in \mathcal{F}$ *map* to labels $L \in \mathcal{L}$ and software blocks $b \in \mathcal{B}$. Thereby it is common that a single function f_a maps to multiple software blocks and labels that implement the function. The edges towards the software model $\mathcal{E}_{\mathcal{L}\cup\mathcal{B}}^{\mathcal{F}}$ are interpreted as an *implementation*. Conversely, the edges in $\mathcal{E}_{\mathcal{F}}^{\mathcal{L}\cup\mathcal{B}}$ can be read as a label or software block *is part of* a function.

2.1.2.2 Mapping between Software Model and Task Timing Model

The link between the data flow in the software model and the timing behavior described by the task timing model are mapping relations between software blocks and tasks. In the graph syntax of the CLM, these mapping relations are bidirectional arcs. The semantic of a mapping arc in $\mathcal{E}_{\mathcal{T}}^{\mathcal{B}\cup\mathcal{L}}$ can be read as "a software block/label is implemented in the task it maps to". In the opposite direction $\mathcal{E}_{\mathcal{B}\cup\mathcal{L}}^{\mathcal{T}}$, the edges create a correspondence between the timing behavior a task describes with the software blocks and labels. A task as the timing instantiation now either describes the timing behavior of the software blocks that are implemented by the task or the timing behavior of a transmission of a label across a task event chain. Hence, the mapping allows to describe the timing behavior of the data flow modeled by the software model. The assumption made here is that each job of a task that maps to a software block which produces a label, creates a **label instance**. It is assumed that this happens with a task's output event. Refinements of this mapping assumption are possible. For instance if the software blocks are runnable entities as described by AUTOSAR. In this case, creation events of the label instances can occur before the output event of the task. [AHE16] presents a

timing analysis for this setup. Similar to the analysis presented in [AHE16] the creation events of label instances can be mapped to the response times for each runnable entity. **Note:** Here we restrict to situations where the creation event is the termination event of a task.

Further, a register based communication is assumed in the timing model, implying that label instances are not buffered.

For labels, two mapping options exist. First, labels can be unmapped. This implies that creation of their instances as well as reading them happens on the same timing resource as the producing task. The reading task in this case simply samples the current label instance. Hence, which label instance is being read by a particular job $\tau_{i,j}$ depends on its activation trace and the termination trace of the task producing the label instance.

Second, labels can map to more complex timing model structures. These structures are chains of tasks with event precedence in the timing graph $\mathcal{TG}$. To account for middleware that influences the timing, four possible mappings of a label on a task chain exist:

- task chains w/o sampling
- task chains w/ source sampling
- task chains w/ sink sampling
- task chains w/ source and sink sampling

Sampling is required if either the producing task, i.e. the task implementing the producing software block, the reading task, which is the task implementing a reading software block, or both are not the source and/or the sink of the task chain. In these cases, sampling-based communication between the tasks is assumed.

Fig. 2.3 shows examples of the four possible mapping options. Mapping a label to a task chain is for instance applied if a label is communicated via an Ethernet stream. In the timing model, the stream consists of multiple event dependent tasks on the resources that model the traversed switch ports where arbitration takes place. If middleware is used, e.g. a network stack, its timing behavior can be captured by a separate task. Depending on the implementation, either source, sink or both may or may not use a network stack.

The “unrolling” of the timing behavior of the data flow defined in the software model yields another perspective on a system: The timing path of data in a system. While an alternating chain of software blocks and labels is only

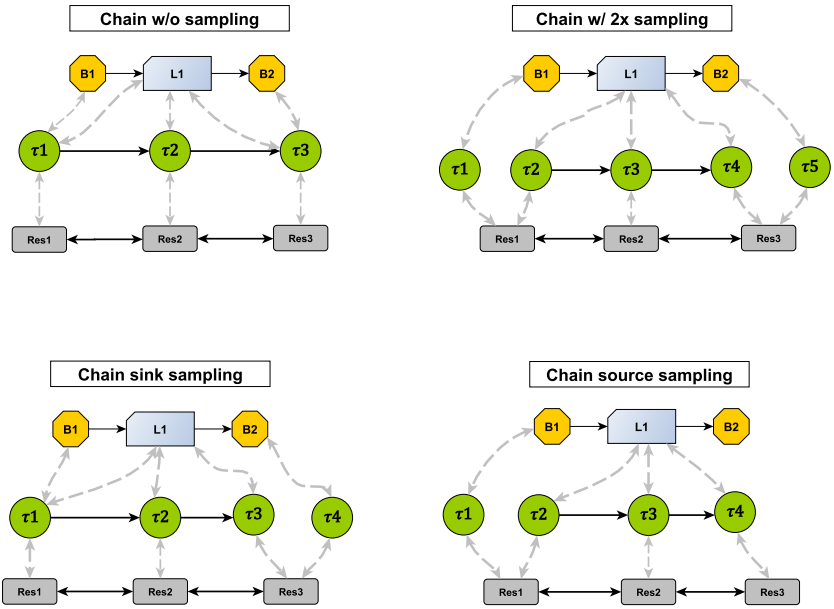


Figure 2.3: Four mapping options of a software model label to task chains

an abstract and implementation agnostic representation of the data flow, the mapping result also covers timing behavior of the implementation. The mapping from the software model to the timing model results in (timing) cause-effect chains for the system.

Definition 2.1.12: BET cause-effect chain

A cause-effect chain $\xi_i = (\tau_0, \tau_1, ..., \tau_n)$ is an ordered set of vertices from $\mathcal{TG}$ such that it maps to a walk $w = (v_1, v_1, ..., v_n)$ in $\mathcal{SG}$ with v_1 mapping to τ_0 and v_n mapping to τ_n.

At run time, instances of cause-effect chains are generated.

Definition 2.1.13: BET cause-effect chain instance

An *instance of a cause-effect chain* $\xi_{i,j} = (\tau_{0,a}, \tau_{1,b}, ..., \tau_{n,z})$ is a sequence of jobs with the property that a succeeding job $\tau_{i,l}$ reads data from its predecessor $\tau_{i-1,k}$. The communication can either be explicit by event dependence if $\exists(\tau_{i,l}, \tau_{i-1,k}) \in \rightarrow$ or sampling based if $\nexists(\tau_{i,l}, \tau_{i-1,k}) \in \rightarrow$.

Note that these sequences are not necessarily deterministic in the BET timing model, as for each producing task, the time when a label instance is produced jitters between the best-case response time (BCRT) and worst-case response time (WCRT). Alike the time a label instance is read can not be precisely predetermined in the BET model. Again only intervals in which a particular job reads data can be computed. This two effects combined leads to the problem that concrete timings of data flow are undecidable in the BET model. However, similar to the response time, possible intervals of data age can be computed as shown by [BDM+17][SMT+18][FRNJ08].

Based on the notion of cause-effect chains, the data-age requirements specified in the software model in Definition 2.1.3 can now be expressed in the timing model. Since label read and write events are directly mapped to events in the timing model, the definition can be interpreted analogously, i.e. on the read and write events in cause-effect chains. Section 2.3 introduces how bounds on these requirements can be computed by applying formal analysis.

2.1.2.3 Mapping between Task Graph and Resource Graph

In order to form a complete timing model, the tasks in the task graph must be related to resources in the resource graph. The mapping direction from tasks to resources captured by the edges in $\mathcal{E}_{\hat{\mathcal{R}}}^{\mathcal{T}}$ can be read as: *a task τ_i mapped to a resource Res_j is implemented on Res_j.*

The edges in the opposite direction $\mathcal{E}_{\mathcal{T}}^{\hat{\mathcal{R}}}$ are interpreted as: a resource Res_i *provides service* to all tasks mapped to it. Beyond Lemma 2.1.11 the mapping relation is further constrained:

Lemma 2.1.14: Task Mapping

The mapping of tasks to resources is constrained such that

$$\rho \subseteq \mathcal{T} \times \hat{\mathcal{R}}$$

is a right-total relation.

$\rho(\tau_i) = Res_j$ hence allocates a task τ_i to exactly one resource Res_j that provides execution time to it according to the resource's scheduling policy. This completes the BET timing model. In summary, it consists of the task graph $\mathcal{TG}$, the set of resources $\hat{\mathcal{R}}$, and the resource allocation function ρ. Formal analysis techniques like compositional performance analysis (CPA) [HHJ+05] allow computing worst and best-case bounds on the response time over all possible traces. Therefore, knowledge about the scheduling policy of each resource is necessary. The WCRT for a task τ_i is denoted by R_i^+ and the BCRT by R_i^-. A more detailed description of how BCRTs and WCRTs can be computed from the specifications of the timing model follows in Section 2.3.

2.2 Hidden Dependency Identification

The first step for dependency analysis is to identify dependencies. While a *dependency* in the CLM can be any walk from a node v_i to v_j in the CLM this simple notion neglects that a number of dependencies are actually accepted or intended. The purpose of the dependency analysis is to identify *unintended* dependencies that e.g. endanger assumptions made for the safety argumentations. Intended dependencies between the architectural views are for instance present if the other view details the implementation. Whether

an implementation is shared e.g. with another function can be detected in the second step – such sharing, e.g. of a software block or a common sensor is the kind of dependencies the analysis should reveal. Hence, in a first step we have to compose the set of model elements between which the dependency is acceptable. Whether an element in a model Y implements an element in model X, can be identified through the mappings with an *implements* semantic. In the case of the presented CLM these are: $\mathcal{E}' = \mathcal{E}^{\mathcal{F}}_{\mathcal{B}\cup\mathcal{L}} \cup \mathcal{E}^{\mathcal{B}\cup\mathcal{L}}_{\mathcal{T}} \cup \mathcal{E}^{\mathcal{T}}_{\hat{\mathcal{R}}}$. To identify the *accepted implementation dependencies* we create an inter-model implementation view of the CLM:

Definition 2.2.1: Inter-Model Implementation View

The *inter-model implementation view* is a Graph $\mathcal{CLM}' = (\mathcal{F} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{T} \cup \hat{\mathcal{R}}, \mathcal{E}')$ containing all model elements, i.e. nodes of the CLM, however the edges set is restricted to edges with an *implements* semantic.

Algorithm 1 extracts the set of nodes between which acceptable dependencies exist by depth-first search (DFS) from a set of source nodes. Suitable source nodes are for instance the node(s) representing a single function or an interconnected function cluster.

Algorithm 1: get_accepted_dependencies($\mathcal{CLM}'$, start_set)

```
Data:
CLM' : Filtered CLM containing only "implementation" edges
start_set : Set of nodes from which mapping dependent elements are obtained
for s in start_set do
    for n in dfs(CLM',s) do
        accepted_deps.append(n)
    end
end
return accepted_deps
```

Besides obtaining the set of accepted dependencies by Algorithm 1, the set can also be directly specified in a model-based development process. Assume an organization where different departments are responsible for the definition and implementation of functions. These efforts can be converged in a joint model base which holds this information without the need of having in-depth knowledge about the efforts of other teams. For instance EEA-ADL supports such development efforts [Vec], [Mat10]. This allows

dependency analysis to avoid costly FMEAs where teams must be brought together [Int18b, Part 2].

Based on the set of model elements between which dependencies are accepted, the CLM can be explored for direct unintended dependencies. Dependencies in the initial sense are any path between nodes of the CLM. In this form they have a transitive nature. In consequence, an enumeration of all possible dependencies would suffer from the path enumeration problem and a possibly unbounded runtime due to an unbounded number of paths. Even restricting the problem to all simple paths, i.e. where a cycle is traversed at most once, does not avoid the problem. Hence, we restrict the search in the CLM to direct neighbours of elements that are in the set of accepted dependencies. Algorithm 2 accomplishes this.

Algorithm 2: find_hidden_dependencies($\mathcal{CLM}$, accepted_dependencies)

```
Data:
CLM : cross-layer model with all inter- and intra-model edges
accepted_dependencies: Set of nodes between which accepted dependencies
exist
hidden_dependencies : Set
interference_nodes : Set
for n in accepted_dependencies do
    for v in get_inbound_neighbours(CLM,n) do
        if v ∉ accepted_deps then
            hidden_dependencies.append((v, n))
            dependence_nodes.append(n)
        end
    end
end
return hidden_dependencies, dependence_nodes
```

The result of Algorithm 2 is a set of nodes at which interference from a non-accepted dependency can be observed (*dependence_nodes*) and a set of edges that led to the insertion of a node in dependence_nodes. In Fig. 2.4 a CLM of a system with two function f_a and f_b is shown. Blue filled nodes show nodes that are accepted dependencies for f_a according to Algorithm 1. Nodes with a red contour are the nodes identified by Algorithm 2 to have hidden dependencies on the set of accepted dependencies returned by Algorithm 1.

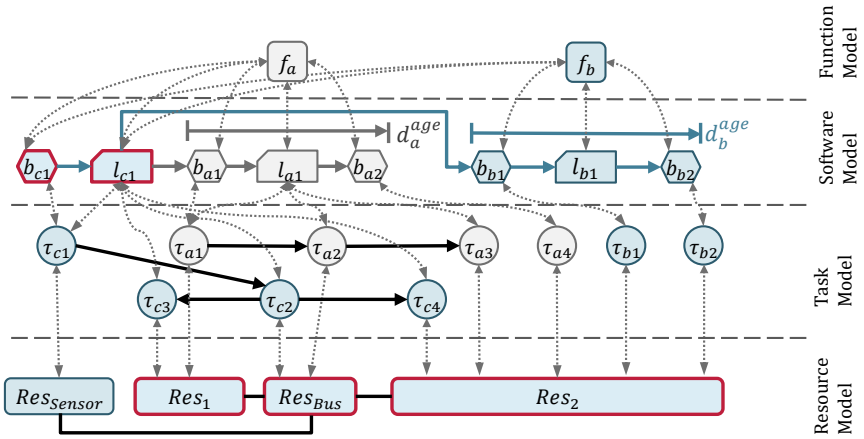


Figure 2.4: CLM of an example system: Blue filled nodes are the accepted dependencies of f_b, model elements with a red contour are nodes with unaccepted dependencies.

As the goal of a design is that it only contains acceptable dependencies, a systematic strategy to investigate dependencies is necessary. Based on the CLM a process with three steps is proposed:

1. Dependency Identification
2. Dependency Quantification
3. Validation of a Dependency

As we have seen, step one can be performed by Algorithms 1 and 2. The rationale behind the second and third step is that if a dependency is present but does not interfere with any requirement such that this is impaired, although unintended by design, the dependency might be acceptable. The example in Fig. 2.4 contains two types of dependencies which, however, might be acceptable:

- common implementation
- resource sharing

W.r.t. the first type, it might be acceptable that the input to software blocks b_{a1} and b_{b1} is from a joint source (b_{c1} that provides L_{c1}). The second type

e.g. happens on Res_2 that is shared by several tasks, however, not all of them are accepted dependencies for f_b.

Each type of dependency requires a suitable analysis to quantify it and evaluate whether the obtained bound is either above or below an acceptance threshold. Fig. 2.5 shows possible actions based on the analysis result. It is noteworthy that there are cases, where a specific direct dependence might be unbounded, but all the transitive dependency paths that traverse it can be bounded sufficiently. In these cases it might be possible to accept a direct dependence that is not bounded, as no unbounded effect can trigger it. However, in such cases it has to be made sure that the element which has a direct dependency does not fail in unsafe fashion and cause interference. For the time being we assume that analyzing and quantifying dependence is possible – in further sections we show how this can be done for timing dependencies due to resource sharing. Assuming that an analysis reveals that a dependency is acceptable we can store this information as an edge property in the CLM. By modifying Algorithm 2 in line 5 to check for this property, a direct dependency can be ignored in the search for unwanted dependencies.

Theorem 2.2.2: Unintended Dependency Acceptance

If each dependency $e = (v_i, v_j)$ on a set of acceptable depedencies A with $v_j \in A$ is acceptable as its extent is below a given threshold no further hidden dependencies exist in the CLM.

Proof 2.2.3:
The proof is by contradiction. Assume that a dependency in the form of a node walk $v_0, v_1, ..., v_i, v_j$ exists, then there must also be an edge $e = (v_i, v_j)$ where the dependency is not accepted. However, if all edges $e = (v_i, v_j)$ with $v_j \in A, v_i \notin A$ such a node walk cannot exist.

□

As an example consider again the system in Fig. 2.4. We first have to consider the dependencies on the software block b_{c1} and the label L_{c1}. The sharing policy in this case is that the data provided by the sensor (L_{c1}) is simply published by b_{c1} without any feedback. Since we assume that none of the sinks is able to exert control over the common source and the software of b_{c1} is implemented to fulfill the highest applicable requirements of either f_a and f_b, we can conclude that the dependencies on the software block

b_{c1} and the label L_{c1} are acceptable. However, if we consider the resources Res_1, Res_{Bus}, Res_2 from the resource model in Fig. 2.4 the situation becomes more challenging. The resources provide service to the tasks, which are mapped to them under a specified scheduling policy. Seen the other way around, the mapping implies that a task's timing behavior is dependent on how much service it receives from the resource according to the scheduling strategy. Yet, the time-sharing behavior and the effect of transitive timing dependencies can be analyzed to quantify the degree of dependence. We put a special focus on timing dependencies. Therefore, we now present how timing behavior can be analyzed to compute bounds on it.

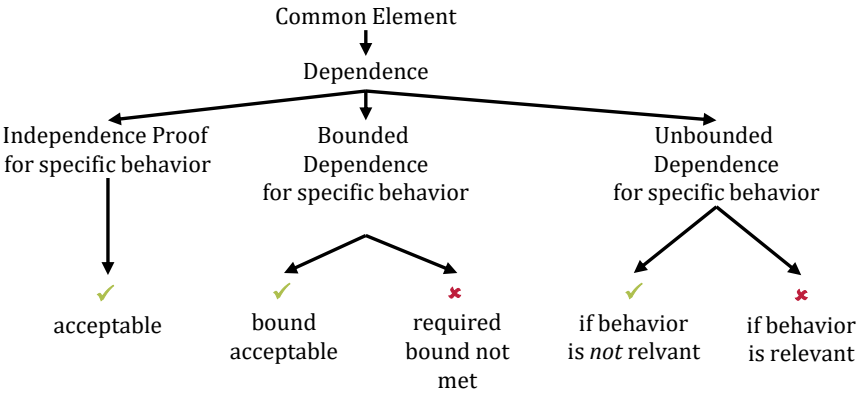


Figure 2.5: Taxonomy of Dependence for the Dependency Analysis Flow

2.3 Analyzing Timing Behavior

The timing behavior of entire systems can be analyzed through different approaches. Here, we resort to the CPA framework due to its compositional nature [HHJ+05]. In CPA the analysis consists of local and global analysis steps. Based on the description of the workload and its parametrization in the task model, several bounds on the timing behavior of a task can be computed. This is done by the local scheduling analysis of a single resource.

2.3.1 The Local Analysis Step

In the CPA framework each resource can be analyzed with a scheduling policy specific analysis. Most of today's analysis approaches emerged from early single resource schedulability tests such as [LL73]. While early work relied on strict periodic tasks and computed schedulability based on device utilization, the focus later shifted towards the computation of response-time bounds under more expressive event models such as periods with jitter [TBW94], [Leh90],[JP86]. Contemporary analysis methods in the end generalized analysis approaches to arbitrary activation patterns [HHJ$^+$05]. All event model based methods have in common that their approach is to compute the largest time interval in which a resource is actively computing jobs. This concept is referred to in the literature as the *busy period* approach. For a conservative analysis, it assumes that all tasks are activated such that they generate the maximum load during the busy period. From all the jobs in the busy period the analysis identifies the job with the longest/shortest response time as the WCRT / BCRT of a task. A generalization jointly reported by Diemer in [Die16] and Axer in [Axe16, Def. 10] defines the timing behavior of a scheduler as a set of functions, consisting of the scheduling horizon function H and the processing functions w^+, w^-. A general overview over the concept is presented in [HAE17]. The processing time functions are defined as:

Definition 2.3.1: Processing Time Functions

The maximum and minimum q-event processing time $w^+(q)/w^-(q)$ return lower and upper bounds on the time interval between the arrival of the first event and the completion of the q-th event for any q consecutive events of task assuming that all q but the first activation arrive within the scheduling horizon of their predecessors.

Whereas the abstract scheduling horizon for any work conserving scheduler is defined as:

Definition 2.3.2: Scheduling Horizon

The maximum q-event scheduling horizon $H(q)$ of any sequence of q events of a task τ is a right half-open interval starting with the arrival of the first and ending just prior to the latest time when a hypothetical $q+1$-st activation would receive ϵ service.

In [Axe16] it is further noted that "The notion of the scheduling horizon tells us whether two events (or rather associated task executions) influence each other timingwise. Naturally, if two events are spaced very far apart, there is no influence. That is the execution of the first has no timing impact on the production of the second. However, if two events arrive very closely (i.e. burst), the processing of the second event is delayed by the first event." In consequence the processing time functions capture dependencies between multiple activations of a task. This notion of intra-task dependencies also allows to compute a maximum number of events q^+ that must be considered to obtain the WCRT, based on the definition of the busy period in [Axe16, Def.13]. This leads to a number of bounds:

Lemma 2.3.3: Worst-Case Response Time Bound

The *worst-case response time* R_i^+ for a task τ_i is upper bounded by:

$$R_i^+ = \max_{\forall 0 \leq q \leq q^+} w_i^+(q) - \delta_i^-(q)$$

Lemma 2.3.4: Best-Case Response Time Bound

The *best-case response time* R_i^- for a task τ_i is lower bounded by the best-case execution time:

$$R_i^- = w_i^-(1) = C_i^-$$

Lemma 2.3.5: Maximum Backlog of Events

The *worst-case backlog* for a task τ_i is upper bounded by:

$$backlog = \max_{1 \leq q \leq q^+} \{\eta_i^+(w_i^+(q)) - q + 1\}$$

In Lemma 2.3.5 $\eta_i^+(\Delta t)$ is the upper event arrival function, which is the pseudo inverse function of $\delta_i^-(q)$. How the conversion can be performed is e.g. given by [Axe16, Eq.3.14 - Eq.3.17] as a revised version of [Sch11, Eq. 3.5 - 3.8].

Furthermore, the reader is also referred to [Axe16, pages 49-50] for the proofs of Lemma 2.3.3, Lemma 2.3.4 and Lemma 2.3.5.

2.3.2 The Global Analysis Step

As tasks can be part of entire task chains in a task graph, important properties of a task are not a-priori known, e.g. the event model. Due to the fact that chains of event dependent task graphs may visit a resource twice, a simple feed-forward analysis where the event model is simply propagated downstream is consequently impossible, since the computation of event models and busy periods become dependent. However, an iterative solution of the problem is possible. Fig. 2.6 shows the analysis flow of CPA that is able to handle this problem. The method of choice in the CPA framework as presented e.g. in [HHJ+05] is that initially unknown event models are initialized with optimistic assumptions at the beginning of the iteration. These event models are then used for the local resource analyzes which consider a single resource in isolation. After the local analyzes for each resource have converged, updated (output) event models are propagated to dependent tasks according to the task graph. How output event models can be computed based on the local analysis results is e.g. given by [Ric05] or [SRIE08]. Either based on response-time jitter, i.e. the difference between worst and best case response time, or in a more sophisticated version based on the computed busy times. The latter leverages that not all out of the q^+ activations experience the WCRT. The iteration over the local analysis step together with event model propagation in the global step is either terminated if non-schedulability is determined in one of the local analysis or if convergence is reached. The global step converges if the propagated event models are stable and no longer change after the local analyzes.

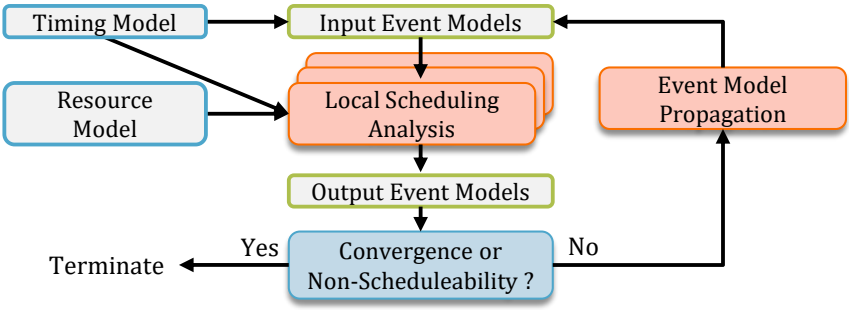


Figure 2.6: The CPA flow based on timing model and resource model. The two steps *local scheduling analysis* and *event model propagation* are shown in orange (Figure according to [HHJ+05]).

After convergence, latencies along task chains spanning multiple resources can be computed from the results. This is e.g. of interest if the latency of a network communication is constrained by an end-to-end timing requirement. A straight forward approach is to sum up the WCRTs along a task chain. Here, [Sch11] presents a more sophisticated approach than summing up the individual response-times. Also, more scheduler specific effects can be leveraged to better bound local response times of path elements and thus end-to-end timing, e.g. in [ATED14] and [TAE15].

2.4 Timing Dependence Graphs

So far we have seen how dependencies on shared elements like computation or communication resources can be identified by dependency analysis. More precisely, the dependency identification in Section 2.2 reveals nodes in the set of accepted dependencies at which the paths of interference "enter" the sub-graph that is the set of accepted dependencies in the CLM. Furthermore, the timing analysis methods from the previous section allows bounding timing behavior based on a number of model assumptions and parameters of the task model and the resource model.

The objective of timing dependency analysis is twofold. Besides analyzing the degree of dependence it specifically has the intent of revealing the timing dependencies of timing requirements formulated in the CLM. Such timing dependencies between functionally uncorrelated parts of a system also manifest in the timing model. Dependencies that, e.g. influence a maximum data-age requirement or a maximum controller dead-time requirement can be hidden on the timing layer. To systematically unravel such dependencies on the timing layer, we introduce a data structure to formally capture the dependencies. This data structure is referred to as the timing-dependence graph (TDG). The idea and concept of the TDG was first published by me in [ME18] and later refined in [MSE18a].

Definition 2.4.1: Timing Dependence Graph

A *Timing Dependence Graph* is a graph $\mathcal{TDG} = (\mathcal{V}^{TDG}, \mathcal{E}^{TDG})$ consisting of nodes $v_i, v_j \in \mathcal{V}$ and edges $e_k \in \mathcal{E}$ where each directed edge $e_k = (v_i, v_j)$ describes that v_j is dependent on v_i. The set of nodes $\mathcal{V}$ consists of task parameters $p \in \mathcal{V}^{Parameter}$ and (intermediate) timing analysis results for a task $r \in \mathcal{V}^{Results}$. Particularly $\mathcal{V}^{Results}$ is dependent on the scheduling policy of the resource of a task $\rho(\tau_k)$.

The rationale behind the TDG is to capture dependencies between timing and resource model parameters on the one hand and timing analysis results on the other. Through a systematic understanding which changes, to either a parameter or a result, influence other parameters or results, the hidden dependencies become traceable paths in the TDG.

To convert parameters and results from the task model in nodes of the respective type we define two conversion functions:

Definition 2.4.2: Conversion Functions

The *parameter conversion function* is a function:

$$\vartheta_p : \mathcal{T} \times \mathcal{V}^{Parameter} \mapsto \mathcal{V}^{TDG}$$

that maps each input parameter type $p \in \mathcal{V}^{Parameter}$ for a task $\tau_i \in \mathcal{T}$ to a node $v_a = \vartheta_p(\tau_i, p)$ with $v \in \mathcal{V}^{TDG}$ and the *results conversion function*

$$\vartheta_r : \mathcal{T} \times \mathcal{V}^{Results} \mapsto \mathcal{V}^{TDG}$$

that maps each result type $r \in \mathcal{V}^{Results}$ for a task $\tau_i \in \mathcal{T}$ to a node $v_b = \vartheta_r(\tau_i, r)$.

In general, four steps are necessary to construct the TDG. These are indicated by color in the TDG construction example in Fig. 2.7. First, for each task in the task graph, the TDG is populated with the nodes describing its parameters and results according to the conversion functions. In Fig. 2.7 result nodes are filled in light yellow and parameter nodes are filled in dark blue. In the second step, all explicit timing dependencies from the task graph between tasks on different resources are added as edges in the TDG.

Task event dependencies on the same resource can be handled by specific scheduling analysis as e.g. described by [SE16] and are handled by step three.

Theorem 2.4.3: TDG Edges for Event Model Propagation

For each pair of dependent tasks $e = (\tau_a, \tau_b) \in \mathcal{E}^{\mathcal{TG}}$ in the task graph $\mathcal{TG}$ with $\rho(\tau_a) \neq \rho(\tau_b)$ dependecy edges $e_k = (v_i, v_j)$ and $e_l = (v_m, v_n)$ are added in the TDG such that:

$$v_i = \vartheta_r(\tau_a, \delta^-_{out}) \quad \wedge \quad v_j = \vartheta_p(\tau_b, \delta^-_{in})$$
$$v_m = \vartheta_r(\tau_a, \delta^+_{out}) \quad \wedge \quad v_n = \vartheta_p(\tau_b, \delta^+_{in})$$

Proof 2.4.4:
The dependencies follow the rationale for the global analysis step of CPA as provided in [Ric05].

□

The example in Fig. 2.7 depicts e_k and e_l for tasks τ_a, τ_b in bold red.

This is followed by the third step which deals with the dependencies on each resource and contains two sub-steps. It adds dependency edges according to the construction of the busy times (w^+ / w^-), which is scheduler specific, and the computation of response times (R^+ / R^-) according to Lemma 2.3.3. The edges for construction of the busy times are shown in black in Fig. 2.7 and correspond to Theorem 2.4.13, the edges for computing the worst-case response time are shown in green. Note that the BCRT nodes and edges are omitted in the figure for clarity, since the BCRT nodes do not have outgoing dependency edges. For the WCRT dependency edges are added to the TDG as follows:

Theorem 2.4.5: TDG Edges for WCRT computation

According to Lemma 2.3.3 edges $e = (v_k, v_l)$ are added to the TDG:

$$\forall\, v_k \in \{\vartheta_r(\tau_i, w^+), \vartheta_p(\tau_i, \delta^-_{i,in})\} : v_l = \vartheta_r(\tau_i, R^+)$$

Proof 2.4.6:
The dependencies follow the equation in Lemma 2.3.3. A proof for the dependencies captured in the formula is provided by [Axe16, Theorem 1].

□

Theorem 2.4.7: TDG Edges for BCRT computation

According to Lemma 2.3.4 an edge $e = (v_k, v_l)$ is added to the TDG:

$$v_k = \vartheta_p(\tau_i, C^-) \wedge v_l = \vartheta_r(\tau_i, R^-)$$

Proof 2.4.8:
The proof is trivial, as events of a task can never be processed faster than the best-case execution time of the task.

□

The second part of the third step must be carried out for each resource individually, respecting its scheduling analysis. As particularly the function w^+ is scheduler specific, transformations for each scheduling policy are necessary. This will be detailed for different policies in the following. Finally, the fourth step deals with capturing the dependencies that influence the computation of the output event model, based on the resource-analysis results and the applied propagation strategy to bound them. W.l.g. we assume busy-window propagation as described by Theorems 1 - 3 in [SRIE08]. The corresponding dependencies are shown in purple in Fig. 2.7.

Theorem 2.4.9: TDG Edges for Output Event Models

For the nodes $v_l = \vartheta_r(\tau_i, \delta^-_{out})$ and $v_n = \vartheta_r(\tau_i, \delta^+_{out})$ edges $e_a = (v_k, v_l)$ and $e_b = (v_m, v_n)$ are added:

$$\forall\, v_k \in \{\vartheta_r(\tau_i, w^+), \vartheta_p(\tau_i, \delta^-_{in}), \vartheta_r(\tau_i, R^-)\}$$
$$\forall\, v_m \in \{\vartheta_r(\tau_i, w^+, \vartheta_r(\tau_i, R^-), \vartheta_p(\tau_i, \delta^+_{in})\}$$

Proof 2.4.10:
The proof idea for the dependencies of $\delta^-_{i,out}$ is that "the distance between any n events at the output can never be smaller than the minimum time between the production of an event m and the production time of an event q that has been produced $n-1$ events earlier". This is proven in Theorem 1 and 2 in [SRIE08].
The proof for $\delta^+_{i,out}$ follows a similar rationale, however with the difference that the 0-th event cannot be produced before $w_i^-(1) = R_i^-$. The full proof can be found in [Sch11].

□

In the following we now elaborate on the dependencies different schedulers introduce between tasks based on the processing time functions of different schedulers. The processing time functions are necessary to compute timing bounds based on the theorems introduced in Section 2.3.

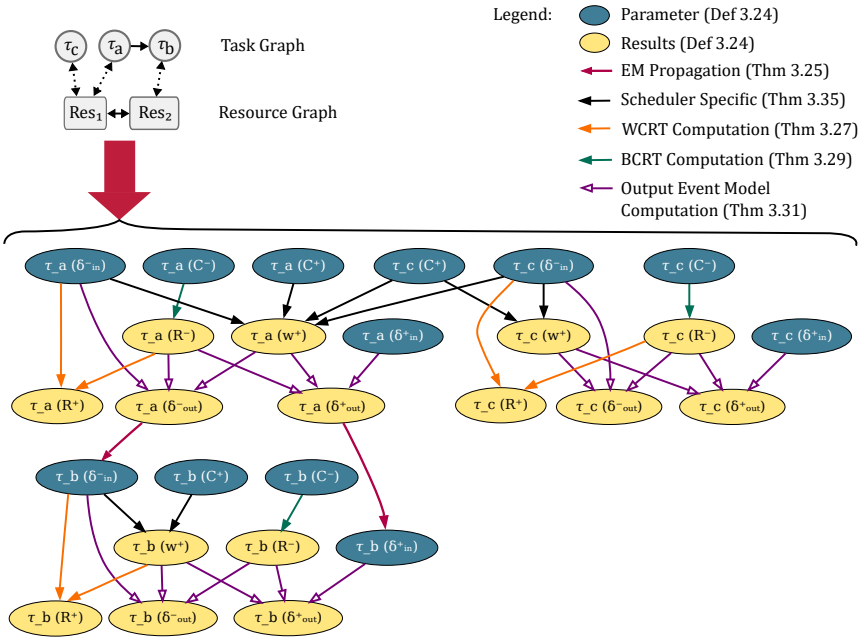


Figure 2.7: Transformation of task and resource graph (top) to its TDG (bottom). The task and resource graph contains three tasks on two resources scheduled by SPP. τ_c has a higher priority than τ_a. The legend on the top right explains which theorems cause wich edges.

2.4.1 SPP

We demonstrate which dependencies a SPP scheduler introduces. In this policy, each task has a priority level assigned. In this policy, at any given time, the active jobs of the task with the highest priority is allowed to execute. The jobs are processed in the order of their activation.

Lemma 2.4.11: Processing Time Function for SPP

The maximum *multiple-event processing time for a SPP scheduler* (neglecting context switch overhead) is upper bounded by:

$$w_i^+(q) = q \cdot C_i^+ + \sum_{j \in hp(i)} \eta_{j,in}^+(w_i^+(q)) \cdot C_j^+ \tag{2.3}$$

with:

- $hp(i)$ being the set of all higher priority tasks mapped to the same resource as τ_i
- C_i^+, C_j^+ being the WCET of task τ_i, τ_j respectively
- $\eta_{j,in}^+(\Delta t)$ being the maximum number of events of a task τ_j in any half-open interval of time Δt

Proof 2.4.12:
For a detailed proof, the reader is refered to [TBW94] or [Sch11]. The proof idea is that on each priority level i the "own" $q \cdot C_i^+$ must be processed plus the accumulated workload of all higher priority task. □

Although w^+ appears on both sides of the equation, it can be solved as a recurrence relation [JP86]. The mathematical properties behind this are in-depth discussed in related work such as [Sch11][TBW94][Leh90].

From Lemma 2.4.11 we can conclude that the processing times of a task τ_i depends on its own WCET C_i^+ as well as all C_j^+ and activation patterns $\delta_{j,in}^-$ of all tasks $\tau_j \in hp(i)$ Note that $\delta_{i,in}^-$ is the corresponding function to $\eta_{i,in}^+$ [Sch11, Eq. 3.5. - 3.8] – it is used interchangeably with $\eta_{i,in}^+$.

Theorem 2.4.13: TDG Edges for SPP

According to Lemma 2.4.11 for each task τ_i on a resource with SPP scheduling, edges $e = (v_k, v_l)$ are inserted in the TDG

$$v_k = \vartheta_p(\tau_i, C^+) \wedge v_l = \vartheta_r(\tau_i, w^+) \tag{2.4}$$

$$\forall j \in hp(i) \; : v_k = \vartheta_p(\tau_j, C^+) \wedge v_l = \vartheta_r(\tau_i, w^+) \tag{2.5}$$

$$\forall j \in hp(i) \; : v_k = \vartheta_p(\tau_j, \delta_{in}^-) \wedge v_l = \vartheta_r(\tau_i, w^+) \tag{2.6}$$

Proof 2.4.14:
For each term on the right side of Eq. (2.3) an edge to the node representing the left-hand side is added thus capturing all the dependencies.

□

This procedure is applied for all tasks on the resource, finally adding all timing dependencies on the resource. The dependencies captured by this step are illustrated by the solid black edges in Fig. 2.7.

The processing time calculation given in Lemma 2.4.11 is the baseline bound for SPP schedulers. It is, however, possible to model and analyze more complex setups. Particularly, shared resource blocking can be considered. As [NSE09] and [SNE09] elaborate, Eq. (2.3) can be modified to consider blocking times. The additional terms also capture dependencies and are inserted in the TDG according to the rationale of Theorem 2.4.13. They are omitted here for clarity. Exemplarily, the following static-priority non-preemptive (SPNP) case demonstrates how blocking is covered in the TDG structure.

2.4.2 SPNP

Besides the common SPP scheduling, also SPNP scheduling is widely spread. The main difference between SPP and SPNP is that once a job receives service, it runs until completion. No preemption, i.e. context switches, can appear within one job. For instance bus arbitration of Controller Area Network (CAN) follows this scheduling scheme as well as the arbitration in Ethernet switch ports if IEEE 802.11Q priorities are used. Besides being common for message scheduling, SPNP is also a supported scheduling mode of the AUTOSAR Classic OS [AUT19b].

SPNP's processing time function is given by:

Lemma 2.4.15: Maximum Processing Time Function for SPNP

The maximum *multiple-event processing time for a SPNP scheduler* is upper bounded by:

$$w_i^+(q) = q \cdot C_i^+ + \sum_{j \in hp(i)} \{\eta_{j,in}^+(w_i^+(q) + t_{cycle}) \cdot C_j^+\} + B_i^+ \quad (2.7)$$

with:

- $hp(i)$ being the set of all higher priority tasks mapped to the same resource
- C_i^+, C_j^+ being the WCET of task τ_i, τ_j respectively
- $\eta_{j,in}^+(\Delta t)$ being the maximum number of events of a task τ_j in any half-open interval of time Δt
- t_{cycle} is a resource dependent constant that denotes the bittime or cycletime to consider boundary effects
- $B_i^+ = \max_{\forall j \in lep(i)} C_j^+$ is the maximum blocking a task can experience by lower and equal priority tasks ($lep(i)$)

Proof 2.4.16:
A proof for Lemma 2.4.15 is e.g. provided in [DBBL07] which analyzes CAN.

□

Due to the additional blocking term B^+ in Eq. (2.7) we add it as another possible result node type in the TDG. Apart from that, the dependencies in Lemma 2.4.15 are similar to the ones in Lemma 2.4.11:

Theorem 2.4.17: TDG Edges for SPNP

According to Lemma 2.4.15 for each task τ_i on a resource with SPNP scheduling, edges $e = (v_k, v_l)$ are inserted in the TDG

$$\forall j \in hp(i) : v_k = \vartheta_p(\tau_j, C^+),\ v_l = \vartheta_r(\tau_i, w^+) \quad (2.8)$$

$$\forall j \in hp(i) : v_k = \vartheta_p(\tau_j, \delta_{in}^-),\ v_l = \vartheta_r(\tau_i, w^+) \quad (2.9)$$

$$v_k = \vartheta_p(\tau_i, C^+),\ v_l = \vartheta_r(\tau_i, w^+) \quad (2.10)$$

$$\forall j \in lep(i) : v_k = \vartheta_p(\tau_j, C^+),\ v_l = \vartheta_r(\tau_i, B^+) \quad (2.11)$$

$$v_k = \vartheta_r(\tau_i, B^+),\ v_l = \vartheta_p(\tau_i, w^+) \quad (2.12)$$

Proof 2.4.18:
The proof how task parameters influence the processing times in SPNP is provided in [DBBL07] for Eq. (2.7). For each term on the right side of Eq. (2.7) an edge to the node representing the left-hand side is added thus capturing all the dependencies. Eq. (2.11) and Eq. (2.12) capture the dependencies for computing the blocking B_i^+ as specified in the remainder of Lemma 2.4.15.

□

2.4.3 Adding further Timing Parameters to a TDG

Up to this point only the dependencies for the output event model and the WCRT have been added to the TDG, since they are the most common bound for a task and are integral part of the CPA analysis methodology. However, there exists an extensive amount of work, how, based on the CPA methodology, further more application specific bounds can be computed. The definition of the CLM already introduced data-age requirements for cause-effect chains in the software model. Whether such requirements can be fulfilled by a BET implementation, can be checked by the analysis presented in [SMT+18]. Hence, another result node type for data age and reaction time for sampling-based task communication on a resource can be added to the TDG node set. Based on Equations 4 and 5 from [SMT+18] dependency edges for the bounds of data-age and reaction-time can be added.

2.5 Timing Dependency Analysis of Systems

The TDG itself captures dependencies between parameters of the timing model. This allows to study (timing) dependencies between functionally unrelated parts of a system that we can find by path analysis in the CLM. Consider again the example in Fig. 2.4. Via the mechanism of the TDG, the timing dependencies on all three resources (indicated in red in Fig. 2.4) that constitute a dependency between f_a and f_b can be studied for any possible scheduling algorithm with all possible scheduling parameter assignments. Through the TDGs for different configurations (e.g. priority assignments) the transitive dependencies become transparent.

Dependency analysis can now be embedded into the design process proposed by safety standards such as ISO 26262 or the more generic IEC 61508 as an

inductive system assessment. Particularly in the phase of the *technical safety concept* [Int18b, Part 4] that ensures the adherence of the implementation to the functional safety concept, this method can be useful.

For an application where the timing has a critical influence on functional correctness (cp. Section 1.1) the technical safety concept must include requirements on timing properties such as response times, data ages, or reaction times. An ASIL requirement on such a parameter denotes that violating the timing property results in a failure which can endanger the safety of the system. There are two aspects to such an ASIL assignment in the technical safety concept: First, the level of residual risk of failure that is tolerated for the requirement, and second that freedom from interference from lower levels must be ensured. The reason for the latter being that freedom from interference is defined as the absence of cascading failure [Int18b, Part1 , Clause 1.49]. Since by definition of SILs / ASILs, failures for lower ASILs are more likely to occur, cascading failure from lower to higher levels must be prevented.

We therefore introduce the property of strict non-interference w.r.t. timing:

Definition 2.5.1: Timing Independence

Two timing parameters represented by nodes $v_a, v_b \in \mathcal{V}$ are *strictly non-interfering* if there exists no node walk $w = (v_a, \cdots, v_b)$ within $\mathcal{TDG}$.

However, it becomes evident in already small examples, that this property is hard to achieve (cf. Fig. 2.7). At this stage the quantification step in the timing dependency flow becomes necessary. Therefore, the requirements as well as the model parameters require a quantification. As safety requirements are basically the accepted occurrence probability of a particular risk or failure they are numerically hard to capture. Hence, many safety standards resort to discritization, e.g. ISO 26262 in the QM level and ASIL A up to ASIL D, IEC 61508 in SIL 1 to SIL 4, or DO-178C with Design Assurance Level (DAL) ranging from E (lowest) up to A (highest). While terminology and proposed methods differ in the details, most safety standards are in many ways coherent in their motivation and how a sufficient level of functional safety can be achieved. We therefore abstract from a particular definition of ASIL / SIL requirements definition and resort to the general notion of a *confidence requirement* for timing parameters:

Definition 2.5.2: Confidence Requirement

The *confidence requirement* is a non-negative integer value, where higher values indicate more stringent requirements, and zero denotes no safety-relevant requirements (i.e. best effort/quality managed (QM)) for timing parameters. The confidence requirement of a TDG node v_i is denoted by $\Gamma_{req}(v_i)$:

$$\Gamma_{req} : \mathcal{V}^{TDG} \rightarrow \mathbb{N}_0^+$$

This definition is compatible with the conception of a safety integrity level in [Int18b] and [The10], as well as the concept of DAL in DO-178C/ED-12C for airborne software systems. Note that DALs are ordered from A (highest) to E (lowest), this can also be mapped to Definition 2.5.2 with Level E being represented by a confidence requirement of 0.

W.r.t. the implementation of safety critical software, standards like ISO 26262 or IEC 61508 dictate certain process measures for design and implementation, increasing the level of care in the process with the required SIL. We thus assume that the efforts of obtaining real-time parameters like the WCET / BCET also increases with this process. This also reflects the first purpose of an ASIL in the technical safety concept: reducing the level of residual risk. In this case the risk is that a wrongful specification via a dependency influences timing parameters with more stringent requirements, e.g. a (computed) response time bound.

Again to abstract from a specific safety standard and process we measure the level of quality as the *confidence* into a parameter of the cross-layer model.

Definition 2.5.3: Timing Parameter Confidence

The *confidence* into a timing parameter is a non-negative integer that discretizes the likelihood that a parameter will violate expected model behavior at run-time. It scales identical to the confidence requirement from Definition 2.5.2.

We assume that the confidence can only be specified if a timing parameter can be analyzed in isolation, i.e. it does not depend on any other timing parameter. These parameters are referred to as *input parameters* and can be obtained from the specification of the designer in the CLM. As input parameters do not depend on any other parameter, they can be easily identified in a TDG.

The set of input parameters I are all nodes which do not have a predecessor in $\mathcal{TDG}$:

$$I = \left\{ v_i : \nexists \, (v_j, v_i) \in \mathcal{E}^{TDG} \text{ with } v_j, v_i \in \mathcal{V}^{TDG} \right\} \tag{2.13}$$

Typically, the input parameter in I are BCET and WCET as well as input event models of non-event dependent tasks.

Note that this notion of confidence into a timing parameter substantially differs from the way MC scheduling annotates execution time bounds. In the MC scheduling model, the theory assumes that the WCET bound changes with the operation mode of the system at runtime, i.e. in which criticality mode a task is scheduled. While Vestals initial paper on the MC model in [Ves07] assumes more than two modes of criticality, most work on MC scheduling only considers two modes - high and low criticality [BD19]. The crucial difference is that the confidence into a parameter annotates the rigour with which the parameter was obtained. The confidence is also a likelihood [1] that expresses the (discretized) probability of misspecification, i.e. how likely it is that a parameter specification will be violated at runtime. The confidence is derived from the quality of a validation process as needed to meet the design requirements of a safety-critical function. Therefore, the required confidence and, hence, the selected value depends on the function criticality which never changes at runtime. This definition follows the argumentation put forth in [EBTLR12]. On the other hand, MC scheduling requires the designer to specify WCET bounds for each task per criticality level *of the system*. In the analyses and scheduler designs presented in the literature these values are used dependent on the current criticality mode of the system, which turns value selection into a dynamic scheduling decision [BD19]. This also implies that response-time bounds depend on the operation-mode change sequences of a system.

The notion of a specified confidence into input parameters and the dependencies in the TDG can subsequently be used for an interference analysis. Therefore, we first introduce a *confidence function* that allows to derive the confidence of an arbitrary TDG node. The rationale guiding this is that the analysis must be conservative in nature, i.e. whenever a parameter depends on more than one other parameter it receives the lowest among its predecessors in the TDG. This is in line with [Int18b, Part 4, Clause 7.4.2] and

[1] Note that in this thesis probability and likelihood are not distinguished in the sense that probability only refers to aleatoric uncertainties. Since no Bayesian statistical analysis and hypothesis test on parameters is done in this thesis, the terms are used interchangeably.

[Int18b, Part 3, Clause 6.4.2.2] that always the highest ASIL takes precedence. If for a dependency in a TDG no "evidence of coexistence" of two safety requirements exists [Int18b, Part 4, Clause 7.4.2.2], the conservative assumption that the lowest confidence will dominate is appropriate. This results in the following recursive definition:

Definition 2.5.4: Confidence Propagation Function

The *Confidence Function*:

$$\Gamma : \mathcal{V}^{TDG} \mapsto \mathbb{N}_0^+$$

with

$$\Gamma(v_i) = \begin{cases} \min\left\{\Gamma(v_j) | (v_j, v_i) \in \mathcal{E}\right\} & \text{if } v_i \notin I \\ \text{the specified value for } v_i & \text{if } v_i \in I \end{cases}$$

returns for every node $v_i \in \mathcal{V}$ a non-negative integer value that represents the maximum achievable confidence into the parameter or result the node represents.

Assigning a confidence to every node of the TDG allows to compare a TDG's confidence value with the specified confidence requirements (cp. Definition 2.5.2). Besides the very strict Definition 2.5.1 of strict non-interference, we now introduce two different forms of timing dependence on a node as a result of dependency paths:

Definition 2.5.5: Timing Dependence

A timing parameter v_b is exposed to

- *bounded timing dependence* if: $\Gamma(v_b) < \Gamma_{req}(v_b)$
- *unbounded timing dependence* if: $\Gamma(v_b) \geq \Gamma_{req}(v_b)$

The two forms of dependence are a direct result of the recursive property of the confidence propagation function. Due to the recursive and hence transitive assignment of a confidence value, any direct neighbor v_a of a node v_b under consideration has the lowest possible confidence by construction. Hence, we know that there exists no node walk $w = (v_i, ..., v_b)$ in $\mathcal{TDG}$ such that $\Gamma(v_i) < \Gamma(v_b)$, because if the walk w would exist $\Gamma(v_b) \leq \Gamma(v_i)$. As a result, bounded timing dependence for a particular parameter is the

guarantee that only timing parameters with the required degree of rigour can influence a timing parameter. Fig. 2.8 shows the possible options how a

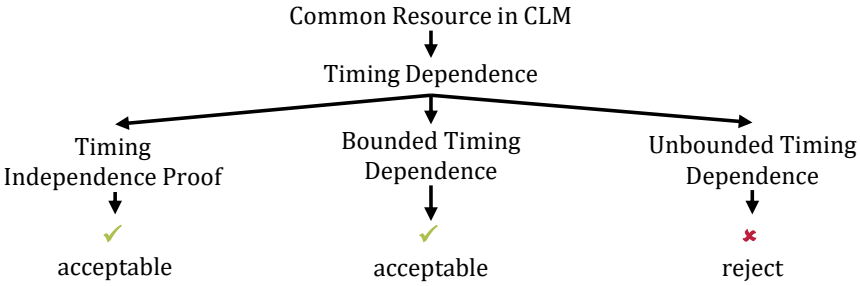


Figure 2.8: Possible options for timing dependencies

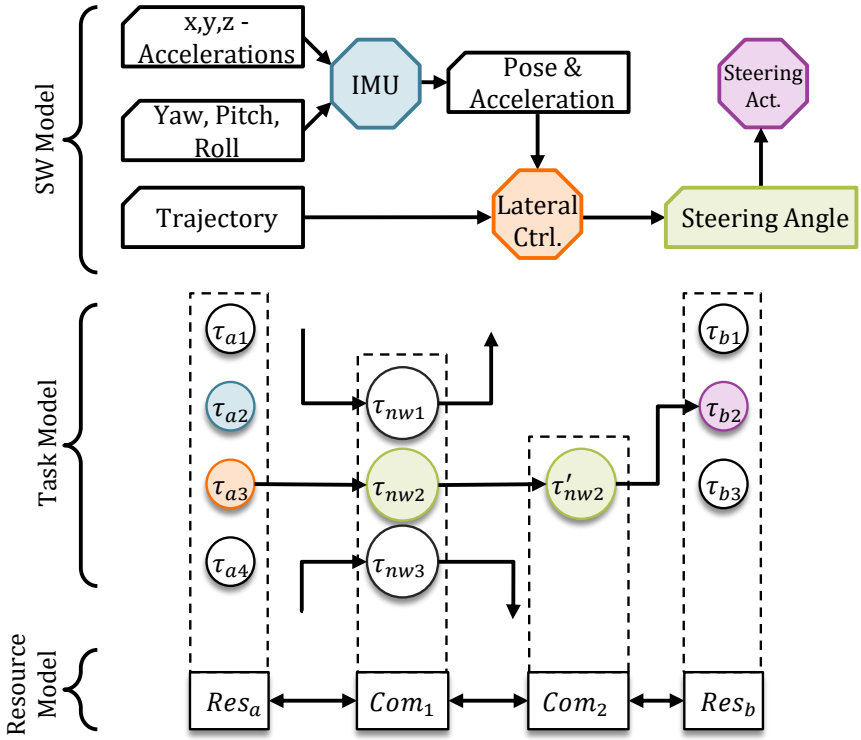
2.6 Case Study and Experiments

2.6.1 Case Study: Lateral Vehicle Control

We consider the lateral controller of a (conditionally) automated research vehicle [Ber15]. For tractability, Fig. 2.9 shows a CLM where the software model is reduced to the essentials of a single cause-effect chain from inertial measurements to the lateral controller. Here, the IMU block bundles inertial sensor data evaluation and pose estimation which is implemented in τ_{a2}. It provides proper-motion and pose estimation to the lateral control algorithm implemented in τ_{a3}. It computes steering angles, which are used by the control actuation. In this implementation, τ_{a3} samples the pose and acceleration data from τ_{a2} and sends them via a network to the steering actuator control which is implemented in τ_{b2}, residing on resource Res_b. Tasks displayed in white in Fig. 2.9 implement other software blocks and may cause unwanted dependencies.

2.6.1.1 Safety Goals

Since the steering function is intended for an automated vehicle, the typical safety-relevant timing properties are the response time of the individual



Mapping Explanation:
- SW Model to Task Model by color
- Task Model to Resource Model by dashed boxes

Figure 2.9: CLM for an implementation of the lateral controller example. The two resources Res_a, Res_b under static-priority preemptive scheduling with priorities descending from τ_{a1} to τ_{a4} and τ_{b1} to τ_{b3}. The communication resources are assumed to be Ethernet switch ports under static-priority non-preemptive scheduling with priorities descending from τ_{nw1} to τ_{nw3}.

tasks, end-to-end latencies and the maximum age of data communicated along the task chain. To identify concrete properties, the ISO 26262 process is applied: It requires the definition of *items* that implement a certain function at the vehicle level, a *functional safety concept* [Int18b, Part 3] to ensure the safety of the items, and a *technical safety concept* [Int18b, Part 4] that ensures the adherence of the implementation to the functional safety concept. We address this by collecting information on functions and items in $\mathcal{FG}$ at the vehicle level. The mappings of the function model [SBM16] provides concrete safety goals for the lateral guidance control of a highly automated vehicle [Ber15]. One of these safety goals (which is the result of an hazard analysis and risk assessment (HARA) process) specifies, that the maximum tolerable overshoot over the reference value is 0.1 m.

The timing bounds in this example can be determined by functional testing of the control algorithm, e.g. with SIMULINK models containing delay and hold elements. These bounds can be propagated via the mapping information to the tasks as model elements in the task model. In the given implementation the maximum data age between τ_{a2} and τ_{a3} and the end-to-end latency from the activation of τ_{a3} to the termination of a subsequent job of τ_{b2} must not exceed given bounds to prevent this. In Fig. 2.10 the pink dashed line depicts the response of the controller under timing errors. An overshoot between 0.5 s and 1 s over the safety goal (depicted as a red line) is clearly visible.

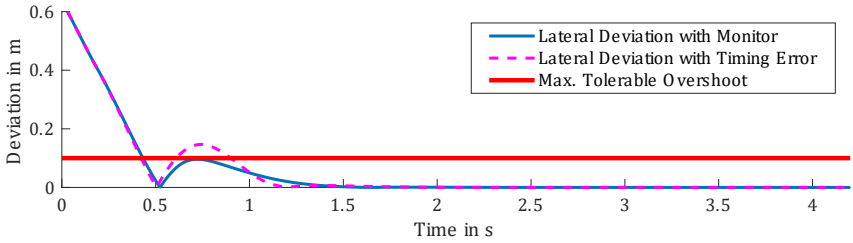


Figure 2.10: Possible lateral deviation under timing errors of the vehicle and maximum tolerable overshoot according to [SBM16]

2.6.1.2 Dependency Analysis

The ISO process now requires to investigate an element's implementation [Int18b, Part 1, 1.32] – i.e. the tasks running the function. Here we can observe, that τ_{a1}, τ_{a4}, τ_{b1} as well as the network communication in τ_{nw1} and τ_{nw2} are

not part of the lateral guidance implementation but share resources with the implementation. Since, the implementation is "shared within a system or among systems of a network" [Int18b, Part 1, 1.129],dependence has to be assumed, unless required independence can be proved.

For the case study we assume that the two resources Res_a, Res_b are scheduled under static-priority preemptive scheduling with priorities descending from τ_{a1} to τ_{a4} and τ_{b1} to τ_{b3}. The communication resources are assumed to be Ethernet switch ports under static-priority non-preemptive scheduling with priorities descending from τ_{nw1} to τ_{nw3}.

With this information, we can generate the TDG for the CLM in Fig. 2.9 but observe that no timing independence according to Definition 2.5.1 can be shown, since paths between nodes belonging to the implementation and timing parameters of other tasks exist in the TDG. Hence, it is only possible to validate the system under bounded dependence according to the taxonomy in Fig. 2.5 and specified in Definition 2.5.5. Note that the bounded dependence check only investigates whether a dependency can in principle be accepted – it however, does not check whether all timing requirements, e.g. WCRTs are met. This is a further quantification step that is necessary.

2.6.1.3 Observations

Through manual analysis of the TDG we can identify the timing parameters of all higher priority tasks as the ones that must receive an equal or higher confidence value for their timing parameters in order to avoid *unbounded timing dependence* according to Definition 2.5.5. More precisely, τ_{a1}, τ_{b1} and τ_{nw1}. Since τ_{nw3} can induce lower priority blocking due to the SPNP scheduling in the network, its WCET parameter also must receive a higher or equal confidence than the implementation of the lateral control cause-effect chain.

Another design option is to assign priorities according to criticality. While this assignment scheme would lead to better results w.r.t. bounded dependence, it can significantly deteriorate the response-time performance of a system which is achieved through rate monotonic (RM) priority assignment [LSD89] [BG03].

While the presented timing dependency analysis method to check for bounded dependence, allows to check any possible combination of platform mapping, priority assignment, and confidence values it does not enable the designer to systematically specify requirements such that it is feasible under

at least bounded timing dependence. This is part of *research question 2 and 3* of this thesis and addressed in the following chapter.

However, the TDG and the timing dependency analysis method allows to perform a systematic and automated FMEA as introduced in Table 1.1. For both process steps *design* and *operation* the potential causes that can trigger the failure mode can be automatically identified, based on the CLM of the system. As we have seen here, these are all the execution time and event model specifications that transitively influence a timing requirement like a response time or data-age. Since, such requirements constitute end-to-end requirements this impact is studied as well. Furthermore, the mapping relations of the CLM allow to reflect all the identified causes on the functional and system level, either triggering redesign, sharpening of specifications/requirements or the introduction of a safety mechanism.

2.7 Related Work

2.7.1 Cross-Layer Modelling

The CLM introduced in this chapter follows the basic principle of *correspondence rules* between different architecture models from ISO/IEC 42010:2011 [Int11], where different layers address different viewpoints on the system. Commonly, the architecture of a system is expressed by architecture description languages (ADLs), which are meta-models with different layers of their own. Such meta-models exist in a wide range of flavors, each addressing peculiarities of domains such as avionics or automotive, but in principle follow common principles that are reflected in the CLM presented in Section 2.1. Inspired by the needs of the avionics domain, AADL was developed as a language to specify avionics and generally dependable systems as part of US funded projects [FGH06][AAD17]. It can be seen as the counterpart to EAST-ADL [1] which was developed and extended in the course of several European projects from 2001 to 2013 [Ass13]. W.r.t. industrial partners it was mainly supported by automotive Original Equipement Manufacturers (OEMs) and Tier 1 and Tier 2 suppliers. EAST-ADL features model layers for the vehicle, functional, and detailed functional level to describe implementation agnostic properties. Its implementation layer, however, relies on AUTOSAR elements and as such is able to express timing requirements

[1] www.east-adl.info

[AUT19a] but not the actual timing behavior of the implementation. Furthermore, a coupling of the EAST-ADL implementation level to the Rubus Component Model exists that allows to separately model control and data flow, which is advantageous for clear specification [BCC+17]. EAST-ADL has also been applied as a modelling basis for functional safety analysis techniques and tools [BDT10] [CDF+14]. It allows functional ASIL allocation with the HiP-HOPS tool [1] which is demonstrated on an anti-lock braking use case. However, the tools and modelling are unable to capture requirements such as timing as technical safety requirements. Their advantage is rather *functional* safety analysis, traceability and consistency management [CDF+14]. Another representative of automotive ADLs is the EEA-ADL [Mat10], which has a number of similarities with EAST-ADL. It was incorporated into the architecture-modelling and requirements-tracing tool Preevision, which was later acquired by Vector Informatik and is developed as part of Vector's Preevision product line [Vec]. All the automotive meta-models have in some form a relation to the AUTOSAR meta model [AUT20] as it is the de facto standard for automotive software. However, as already mentioned AUTOSAR provides a number of liberties for implementers and as such mainly formulates requirements and not the behavior of the implementation. This gap is closed by the presented CLM for timing properties and behavior.

2.7.2 Dependency Analysis on CLMs

W.r.t. cross-layer dependency analysis to reveal hidden dependencies across model layers [BB18] and [BBK19] present approaches that are function centric but do not cover extra-functional properties such as the timing, which is in focus of this thesis. The authors investigate how design parameters of the functional architecture, in the case of [BB18] state-charts, and physical parameters are interdependent. In the concrete use case they investigate the hidden dependency on the current consumption of the hardware of an Adaptive Cruise Control (ACC) function. Similar work is conducted for a buck converter in [BBK19]. Both leverage Preevision / EEA-ADL as a design description and model physical properties which are not captured in EEA-ADL in a Ptolemy II model [EJL+03] [Lee09].

Other authors address the identification of potentially harmful dependencies as a design space exploration (DSE) problem. [SVZ15] sees the problem of hidden dependencies as an integration problem, however the synthe-

[1] https://hip-hops.co.uk/

sis mechanisms proposed for the employed AUTOFOCUS3 framework [1] [VEH14] try to circumvent timing dependencies by exclusively targeting time-triggered systems. The time-division multiplex (TDM) avoids timing dependence, as for TDM scheduling non-interference can be proven. Yet, the time-triggered scheduling and communication results in inefficient resource use and suffers from the problem that for a multitude of applications on a platform sufficient time slots must be assigned. Furthermore, it is not compliant to the AUTOSAR Classic platform [AUT19b] which features SPP scheduling on execution resources. In the field of DSE for multi and manycore systems authors aim for application isolation on these shared platforms through suitable mappings of the applications on the execution platform. Such platforms typically consist of a number of tiles connected by a network-on-chip (NoC); each tile posses one or more cores to process workload. A tile can also host further peripherals such as memory, accelerators or memory interfaces [TAED13]. For instance [PSWT19] presents an approach for finding mappings that can leverage multiple inter-application isolation schemes on the platform. Other authors, for which [PSWT19] also provides an overview, limit their scope to only one isolation scheme for the entire mapping problem. The three inter-application isolation schemes typically considered are:

- core sharing: where timing isolation properties are only based on conservative WCRT analysis
- core reservation: which also provides spatial isolation from other applications on a core, however, interference on peripherals of the tile exist
- tile reservation: which raises spatial isolation of concurrent applications to tile level and enables the largest reduction in the worst-case timing interferences

Yet, as we can observe from the description of these mechanisms, we can see that core sharing effectively has the problems that are investigated as part of the cross-layer dependency analysis for timing, i.e. a shared resource that makes timing mutually dependent unless proven otherwise. Current WCET analysis tools can not deliver this proof for any kind of core architecture [WEE+08]. While the other two seem to be able to execute exclusively on cores and hence do not experience timing interference, this is a false assumption. The timing behavior of both is still dependent on inter- and intra-tile communication, e.g. on-tile buses or the NoC, for which [PSWT19]

[1] https://af3.fortiss.org/

and others use timing analysis (such as e.g. [RE15]) to argue isolation. However, this form of isolation must not be confused with bounded dependence from Definition 2.5.5 as the isolation property requires sufficient confidence into the parameters. While WCETs can be bounded accurately in network transmissions, as they correspond to the duration of bittimes, event-models, i.e. the access patterns are the primal source of uncertainty in the timing analysis. On the tiles' cores the challenge of obtaining WCET's with sufficient confidence remains [WEE+08][Wil18]. It has to be noted here, that [PSWT19] only takes one shared resource into account which is the set of processing resources. Other global effects like access to dynamic random access memory (DRAM) are either neglected or abstracted in the NoC's timing analysis, as in the case of shaping mechanisms for off-tile memory access.

SDRAM accesses are managed by special controllers – their timing behavior, i.e. the worst-case time it takes to serve a memory request from controller to the DRAM, together with the timing behavior of the rest of the memory hierarchy (caches, etc.) significantly influences whether sharing such a resource can performed safely. Safe in this context is that it happens with bounded dependence. For the effects of SDRAM access in real-time systems the authors of [GHPP18] provide a survey comparing nine different publications reporting on designs of memory controllers for real-time systems. The surveyed work is of interest, as standard off-the-shelf embedded platforms might provide no bound on DRAM access latency at all [WKP13]. However, the conclusion from [GHPP18] is that no universally preferable memory controller design for real-time systems exists. Rather, the authors suggest makeing the choice application specific [1]. This implies that a technique is necessary to identity the need for a more rigorous specification, specifically if workloads of different timing criticality are mixed. Such a tool can be found in dependency analysis, if its CLM is extended to cover also memory accesses in a separate model [2] rather than abstracting it, as it is the state-of-the-art in task-level response-time analysis.

The same claim can be made for basically any shared (stateful) resource in a design, as the state when a task accesses the resource influences the execution behavior. The question always is: Is the state predictable or can it at least be bounded in order to bound the timing influence? So far this section has mainly considered shared resources as part of an SoC – however also more macroscopic shared resources play an important role

[1] In this context application refers to the composition of functions and software executed on the particular hardware their desired memory configuration and necessary analytical guarantees.

[2] This is possible, since the CLM's design is flexible w.r.t. the models. An additional model only requires the specification of abstraction and concretization mappings.

in whether end-to-end cause-effect chains are sufficiently independent of harmful interference. As the last representative of such a shared resource, in-vehicle networks are discussed. In-vehicle networks can consist of multiple interconnect technologies (CAN, LIN, FlexRay, Ethernet, ...) with numerous gateways between them. While the real-time community has studied many of the technologies w.r.t. their timing behavior [DBBL07][NNEB12][NNEB12][TE16b][TAE15][TSAE16b], and even suggested dynamic resource broking (which is able to provide guarantees) under changing workloads [Kos20][KSE20], the necessity of determining whether the specification quality actually allows claiming bounded interference remains. Assessing the specification quality to qualify the mechanism is first introduced with timing dependency analysis on top of these mechanisms. This corresponds to the demands made by [EN16], that in order to argue "sufficient independence", the mechanism and hence also the data fed to it need to be sufficiently qualified.

Chapter 3

From Verification to Synthesis

Mixed-critical workloads make design and integration a challenging task for hardware-software co-designers – especially for today's Level 2 and Level 3+ vehicles. This criticality is in the literature often interpreted as the safety relevance of a (timing) requirement. Further, some authors assume that the number of criticality level can be limited to a high and a low criticality level, especially in the context of scheduling theory. There is even the assumption that the higher criticality level can interfere with the lower level in order to carry out its task in normal operation. [BD19] provides an extensive overview over this work. It becomes evident from safety standards that such a behavior is not straightforwardly covered by the design rules of safety standards. Hence, contrarily to the works summarized in [BD19], in [EN16] the authors argue that standards define separation in both ways, meaning that at any criticality level the timing should not be affected by (timing) errors in the other criticality level. A fact that is clearly not considered when scheduling algorithms are allowed to drop low-criticality tasks in favor of high-criticality ones in normal operation mode.

A particular conclusion from [EN16] is that the mechanism which integrates software with different criticality must be qualified to the highest applicable level – especially if it is the mechanism that should guarantee *sufficient independence*. For MC scheduling as presented in [BD19], the scheduler is such a mechanism according to [EN16]. The research presented here is an extension

of this argumentation, in the sense that it quantifies the (in)dependence (cf. Definition 2.5.5).

[EN16] motivates that "usually, safety-critical functions are subject to timing requirements". However, technical safety requirements, e.g. an ASIL, on timing requirements and how they originate are rarely discussed in the literature. Furthermore, contemporary safety cases often lead to designs where the violation of a timing requirement does not contribute to hazard. E.g. in the "E-Gas Konzept" which is a safety concept for combustion engine passenger cars to prevent unintended vehicle acceleration, no timings in the functional layer are considered safety relevant, as it is assumed that timing violations only lead to effects captured by the functional monitoring level (*"funktionalen Überwachungsebene"*) [EGA13]. This is only possible due to the assumption of long fault-tolerant time intervals (FTTIs) [Int18b, Part 1, Clause 3.58] that are significantly longer than the control period, e.g. in the case of "EGAS" the time duration from error detection to start of a reaction is in the order of 500 ms, whereas the control period of the software realizing the function ("Ebene 1") is in the order of a few ms. However, with the advert of higher automation levels, the time to transition into emergency operation becomes more crucial [Int18b, Part 1, Figure 5], as vehicles must be brought to a risk minimal stop. Hence, it can be expected that the transition times into safety operation also receive timing requirements that are subject to a technical ASIL.

While timing dependence and confidence analysis can be valuable tools to asses requirement satisfaction, they also require that confidence requirements (in representation of a technical safety requirement) are known for the analysis (cf. Section 2.5). For illustration, we again consider the initial example of a lateral controller of a (conditionally) automated vehicle. In the case study in Section 2.6.1 we have seen that under a given priority assignment and given confidence requirements only a simple binary verification decision is possible: acceptable or not acceptable under bounded dependence. For the implementation of a safety goal, a systematic derivation of confidence requirements on timing parameters of foreign software is necessary or that methods to increase the confidence of parameters is necessary (if critical software depends on it according to the TDG).

Hence, in the following we address the design flow from a different perspective: First, a mechanism to synthesise confidence requirements for all specified timing parameters based on safety requirements of timing requirements is proposed. Second, we address the question of how the design flow can handle situations where either additional confidence requirements

cannot be imposed or otherwise guaranteed. The former being the case when software is already implemented or the hardware and OS/middleware platform is already chosen, the latter in situations where no sufficiently conservative WCET estimation tool is available for the intended hardware-software platform. This is especially a problem, as the platform is often an unalterable constraint in industrial practice, e.g. due to supply chain decisions.

3.1 Synthesis of Implementation Requirements

In this section the question of up to which level of confidence an input parameter must be obtained is addressed. This is a synthesis step, since it generates a requirement on the necessary level of confidence of an input parameter. This is interesting, since quite an extensive number of publications cling to the misconception that a functional safety requirement's ASIL resembles time criticality of a software component. However, this is not necessarily the case, as not every safety critical function is also time critical.

Section 2.5 has elaborated how timing dependencies of a system can be formally captured in a data structure, namely the TDG. In this graph nodes are timing parameters and each edge denotes the timing dependence of the target on the source of the edge. Further it has been shown how timing requirements such as a WCRT or data-age requirement can be added to a TDG. By investigating the TDG with confidence analysis, nodes in the TDG with a mismatch of achievable confidence and confidence requirement can be identified. More precisely, the dependency which leads to an insufficient confidence in a model parameter can be identified. The result are parameters which lead to an *"illegal"* influence in terms of safety. However, this requires that every node, or at least every root (TDG nodes with in-degree of zero), has a confidence value in its specification. Often in a design, designers do not know how the timing parameters of "their" software need to be qualified. Here the presented confidence assignment algorithm can help, by exploiting the knowledge of timing dependencies.

Let us now assume that confidence values can be freely assigned and this way become a requirement for the specification of the parameter. First, note that the confidence imposed only reflects how faithful the parameter is w.r.t. the rest of the model (in the sense of a scientific model according to [LS18]. Confidence as it is presented here can not easily cover effects beyond the model, e.g. hardware failures that would alter the structure of the

model. Nevertheless, safety engineering requires that for higher criticality such failures must be taken into account. These can be mapped either on parameter behavior, e.g. certain types of hardware errors can be abstracted by longer execution times, or require an additional model. Second, note that if the parameter is dependent on the implementation, it also becomes a requirement on the implementation. For instance to what degree of rigour a WCET parameter must be determined. For this purpose, we only assume that the TDG nodes of (timing) requirements have confidence requirements assigned. These are present in case the particular timing requirement has an implication on (functional) safety as they are derived as technical safety requirement.

Originating from the TDG nodes with confidence requirements, we now backtrack through the graph to initial timing parameters and assign them the highest confidence requirement to which a path exists in the TDG. In Algorithm 3 this is done by reversing all edges in a TDG $\mathcal{TDG}$ and performing either breadth-first search (BFS) or DFS from each requirement and upon visiting a node assigning it the maximum of either an already present confidence requirement or the value of the confidence requirement of the source of the search, i.e. the timing requirement node. By selecting the maximum, a conservative requirements assignment is achieved, as a higher confidence implies a lower probability of specification error. This is in accordance with clause 7.4.2.2 part 4 of ISO26262 [Int18b].

Algorithm 3: assign_conf_reqs($\mathcal{TDG}$, requirement_nodes)

```
Data:
requirement_nodes : the nodes carrying a confidence requirement
TDG : The TDG to work on
reverse_edges(TDG);
for n in requirement_nodes do
    confidence_req = get_conf_req(n);
    for k in dfs(TDG, n) do
        cur_confidence_req = get_conf_req(k);
        new_confidence_req = max(cur_conf_req, confidence_req);
        set_confidence_req(k, new_conf_req);
    end
end
```

Theorem 3.1.1

When Algorithm 3 terminates it has assigned the maximum confidence necessary to each node reachable from a requirement in *requirement_nodes*.

Proof 3.1.2:
This simple proof consists of two steps. First of the inner loop that visits all reachable nodes from one timing requirement node in the TDG and second that for the outer loop a conservative assignment of confidence requirements is made. The proof for the inner loop is by induction. The function *dfs* in line 4 explores the graph recursively. Upon the first visit a node is marked. The recursion terminates and does not return any more nodes because ever call to the recursion it to an unmarked node, and each recursion call marks a node. For k reachable nodes there are k calls to the recursion before it stops.
Suppose that a node w is reachable from n and is not marked when *dfs* terminates. Since w is reachable there is a path $n, v_0, v_1, ..., v_l$ with $v_l = w$ from n to w, and a first node v_i that is not marked. However, this is in contradiction since the recursion function marks v_{i-1} and would have examined v_i via the edge (v_{i-1}, v_i). Consequently, *dfs* visits every node reachable from n.
The proof for the outer loop is now trivial. The outer loop visits every TDG node reachable from every node in *requirement_nodes*. Upon visiting, it assigns the maximum confidence requirement. Even if a node is visited multiple times due to calls to *dfs* from multiple origins, the order from which a node w is visited is irrelevant to the maximum operation.

□

TDG nodes in the unreversed graph that have an in-degree of zero, i.e. are roots, are specifiable timing parameters. All of these nodes have received a confidence requirement if a timing requirement in the TDG depends on it, since the search algorithm (cf. line 4) yields all nodes discovered from the node representing the timing requirement.

The confidence requirements on the input parameters now specifies the degree of conservatism up to which the input parameter must be known for a timing analysis like CPA. In that sense, the confidence is interpreted as a probability of failure of the specification. Since the TDG is constructed based on symbolic evaluation of the busy window, which accumulates the inter-

ference, its edges are the possible propagation paths of such a specification failure. CPA and other analysis methodologies on the other hand assume conservative bounds on its input values, i.e. that they are never exceeded. The interpretation of the confidence requirement as a failure probability, however, allows more flexibility in the specification and verification of the system. It allows performing specification in a less rigorous manner, as it is now permissible to specify timing parameters not entirely conservative but with a certain margin of error. Whether this probabilistic error can have an effect on a timing bound such as response time, can be determined by the path analysis in the TDG. Assume that a timing parameter n of a now given confidence Γ_n only influences parameters with similar or lower confidence. In this case it is permissible that the error of parameter n influences its dependent parameters due to their lower confidence, as the equal or lower confidence means that its error probability is equal or *higher* than the one of n.

3.1.1 Case Study

As an example consider the partial CLM in Fig. 3.1 which is an extension of the example from Fig. 2.9 in which the lateral controller is integrated on a shared platform. The focus here is on an additional function f, which for simplicity is assumed to be implemented by a single software block in τ_{b4}. For safe control performance, the WCRT of τ_{b4} is critical. The system under consideration features two computation resources and communication resources, e.g. two CPUs and an Ethernet network. For computation SPP scheduling is assumed while the communication resources schedule according to the SPNP policy. Further the example assumes that the task with the requirement under consideration resides on the second resource on an intermediate priority, e.g. due to RM priority assignment. A communication stream originates on the first resource towards the second resource, however it is unrelated to the function implemented in τ_{b4}. This communication stream competes with two other traffic streams for a switch port. The competing tasks on the switch port Com_2 are τ_{nw1} and τ_{nw3}. For illustration no other requirements besides the WCRT of τ_{b4} is assumed to have a confidence requirement.

The resulting graph of Algorithm 3 is shown in Fig. 3.2. Note that the figure does not depict the TDG as introduced before but with reversed edges that show the propagation of the requirement as described in this section.

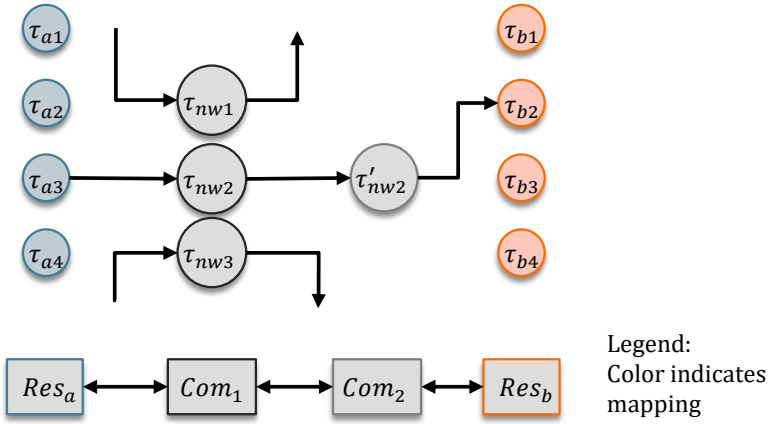


Figure 3.1: CLM for an example system with two computation resources and two communication resources

While it is not obvious from the structural representation of the system in the CLM that a network stream interfering via scheduling has an impact on the timing requirements of f, it becomes obvious from the result of Algorithm 3. The timing parameters of τ_{nw1} that specify its maximum length, i.e. the WCET for network frames, and activation pattern on the switches output port scheduler now would receive a confidence requirement. Besides both network stream – interfering either due to higher priority or due to lower priority blocking – not only tasks executing on the same resource as τ_{b4} can interfere with τ_{b4} but also tasks from Res_a that can influence the network stream although they are functionally unrelated to τ_{b4} and the function f it implements. Algorithm 3 discovers all these parameters systematically, as can be seen in Fig. 3.2.

By applying the algorithm to each node with a confidence requirement the (specifieable) timing parameters receive a confidence requirement such that confidence analysis would not reveal a mismatch for the function under investigation. For correct operation the requirement nodes in the input list of Algorithm 3 must be in ascending order of the confidence requirements.

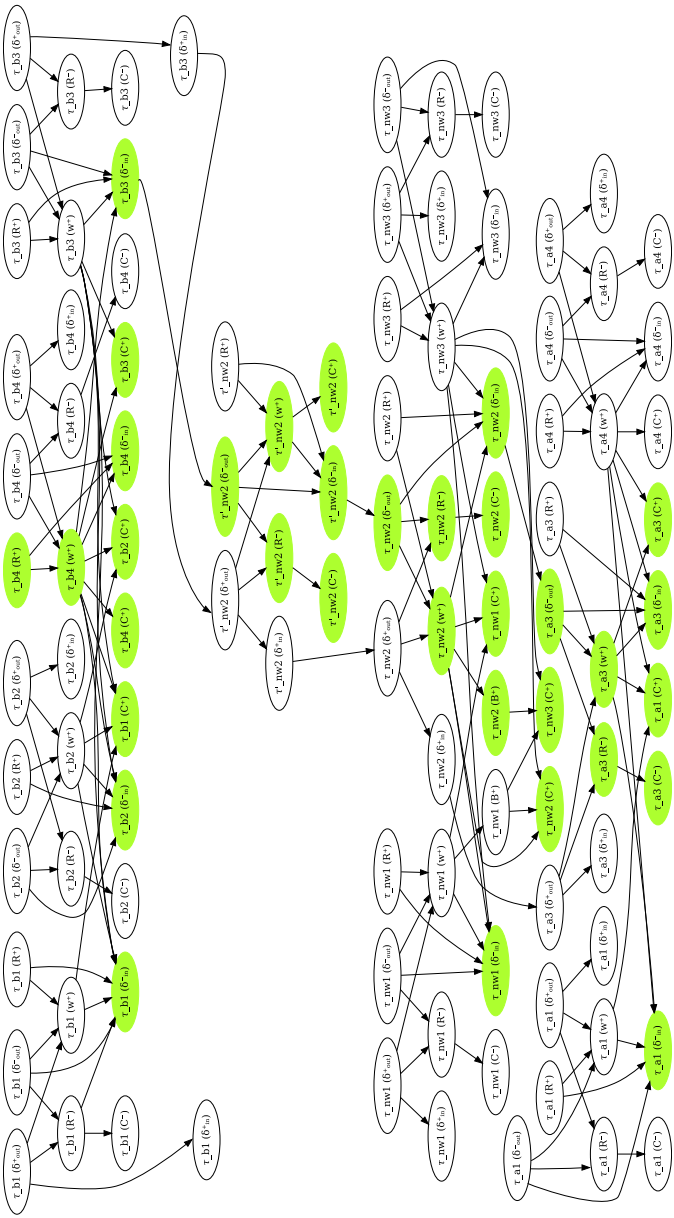


Figure 3.2: TDG with reversed edges for the example system depicted in Fig. 3.1. Colored nodes receive the confidence requirement from the WCRT requirement of τ_4.

3.2 Enforcing Properties at Runtime

In situations where bounded timing dependence or timing independence cannot be substantiated by the confidence analysis, design measures are necessary. The first option is to increase the confidence into the relevant input parameters that cause the deterioration of the confidence below a confidence requirement by improving the qualification process of the parameter as required by the confidence requirement synthesis method presented above in Section 3.1. In industrial scale designs this can, however, be complicated due to the distributed development process across several suppliers. Other reasons are that code development process and e.g. WCET quantification have already been done when the software is integrated into a specific ECU and timing dependence and confidence analysis is conducted as part of the integration process. The second option is to redesign e.g. the task allocation to ECUs and execution cores. While possibly practical for smaller designs it puts enormous stress on other involved system designers, as communication relations change and might additionally interfere with other concerns. E.g. certain software must be executed on resources with the respective peripherals, i.e. are not arbitrarily relocatable. The third option is to use monitors to conform run-time behavior to expected model behavior. A monitor raises the confidence into the parameter, by enforcing behavior of a parameter. In conjunction with the rest of the model, it guarantees valid and acceptable function behavior. The general idea of this third option and how it can be realized is first described in [MSE18a].

In the following we elaborate on different options for monitoring from related work, and their drawbacks and assets.

3.2.1 Related Work – Monitor Types

In the literature three basic types of monitors can be identified to enforce model behavior: The first type are execution-time monitors that enforce properties on execution time. They either prevent exceeding an upper bound such as the WCET or can hold back events, enforcing premature release of an event before the BCET. Basic WCET monitoring is e.g. described in the AUTOSAR OS specification [AUT19b, Chapter 7.7.2]. The second group of monitor types are event model monitors. These resort to enforcing standard event models as they are described for CPA. [NMA+12] comprehensively describes monitoring for arbitrary event models, while other approaches (e.g. [WL96]) limit their scope to periodic event models. Note that periodic

event models are a subset of arbitrary event models. The last group are techniques that directly monitor the induced workload of a task, i.e. they combine execution time and event model monitoring. [NAME13],[NQEL13],[HCBK12] are representatives for this class of strategies. The latter can also be used to conform the convex-hull of an event-arrival curve to a model specification. However, this implies unnecessary conservatism that has also to be considered in the analysis and prevents slack usage by best-effort tasks. None of these related works elaborate on how to apply the respective techniques in the system-level context, i.e. a coordination where to monitor in designs with multiple resources. Neukirchner in [Neu14] partially lifts this restriction, however, assumes that every task that potentially interferes is equipped with a monitor.

Although single monitors can be instantiated with manageable overhead, instantiating a multitude on one resource or across a distributed system can quickly consume valuable system resources, e.g. exclusive timers for monitors. On embedded platforms these can be a scarce resource and thus monitors might quickly consume a system's resources. Consequently, strategies where each and every timing parameter is monitored (as in principle proposed by [Neu14]), quickly becomes too expensive in terms of resource usage.

Furthermore, enforcing all timing parameters also prevents efficient slack usage. More precisely, it prevents best-effort applications to consume spurious workload if monitors enforce optimistically modelled worst-case behavior.

3.2.2 Effects of Monitor Use

The primal effect of a monitor that enforces a defined run-time behavior is that it increases the level of confidence. We only consider monitoring techniques that do not produce false negatives (cf. previous section). Consequently, a monitor will ultimately lift the confidence level of the parameter it monitors and enforces. We therefore modify the *confidence function* from Definition 2.5.4 if monitors are present:

$$\Gamma(v_i) = \begin{cases} \min\left\{\Gamma(v_j) | (v_j, v_i) \in \mathcal{E}\right\} & \text{if } v_i \notin I \\ \text{specified value for } v_i & \text{if } v_i \in I \text{ and not monitored} \\ \text{confidence of the monitor} & \text{if } v_i \text{ is monitored} \end{cases} \tag{3.1}$$

A monitor primarily has an effect on the value it enforces, the modified confidence function conveys the transitive effect on the timing dependencies. Remember that a low confidence in a parameter implies a higher likelihood of a specification fault and that in consequence the run-time behavior of this parameter might deviate from expected model behavior. Since a monitor enforces a specified value, it reduces the ability of a timing fault to cascade at run time. It can already be studied in the second example that it is not imperative to monitor all possible timing parameters to achieve satisfaction of confidence requirements. Assume that the WCRT of τ_a has a confidence requirement of 3 and the WCRT of τ_b of 2. Based on the specified confidences of input parameters, it is easy to conclude, that solely by monitoring and enforcing the WCET C_c^+ of τ_c is sufficient to fulfill the two confidence requirements.

3.2.3 Placement Strategies for Monitors

In the case of the lateral controller example from Section 2.6.1 the ideal placement of monitors is straightforward, as the TDG is humanly tractable. However, for a general application of monitoring in the design flow of distributed systems, a systematic strategy is necessary. In the following, two approaches are presented to systematically place monitors in a distributed system in order to fulfill all confidence requirements of the system. The first presented strategy is a greedy approach, that places monitors on all timing parameters without predecessors that the strategy discovers and have a lesser confidence than required. The second strategy is based on a min-cut of timing dependency (sub-) graphs.

3.2.3.1 Greedy Timing Monitor Synthesis Strategy

In the design of such a strategy one must consider that not all types of nodes in a TDG can be monitored. In case of the greedy approach we will only target nodes for monitor synthesis that are input nodes. Since for all input types ($\{C^+, C^-, \delta_{in}^+, \delta_{in}^-\}$) monitor implementations are available, this fact does not restrain this strategy.

Based on the confidence analysis of the timing dependence graph we can determine for every constraint of the system all input parameters it depends on. More precisely, for every node v_i in the TDG with a confidence requirement we can construct the *reachability graph* $\mathcal{TDG}^{reach}(v_i)$ and determine all roots, i.e. nodes with an in-degree of 0, of it. This yields all input parameter nodes which transitively influence the confidence of the constraint under investigation. We refer to these nodes as the *influencing input nodes*. We subsequently compare the confidence values of these nodes with the confidence requirement of the constraint under investigation. For each input node that is interfering with the constraint node in the strict sense we generate a monitor and lift the node's confidence to the highest possible level. We perform this for all specified constraints, i.e. their corresponding nodes in the TDG.

Theorem 3.2.1: Greedy Monitor Placement

After this procedure the confidence requirement $\Gamma_{req}(v_a)$ of a node v_a is fulfilled.

Proof 3.2.2:
The proof is by contradiction: Assume there exists a node $v_i \in \mathcal{V}^{reach}$ with $\Gamma(v_i) < \Gamma_{req}(v_a)$. In this case a path from v_i to v_a exists propagating the confidence value according to the confidence function reaching v_a.
However, in cases where v_i is an input parameter it has received a monitor contradicting the assumption, or it is a node within $\mathcal{TDG}^{reach}$. In the latter case it must receive its confidence value through evaluation of the confidence function from the reachable input nodes. By construction of $\mathcal{TDG}^{reach}(v_a)$ and $\mathcal{TDG}^{reach}(v_i)$, $\mathcal{TDG}^{reach}(v_i)$'s nodes are a subset of $\mathcal{TDG}^{reach}(v_a)$ and have received monitors if their confidence is below $\Gamma_{req}(v_a)$. In consequence a

node v_i with $\Gamma(v_i) < \Gamma_{req}(v_a)$ that could still influence $\Gamma(v_a)$ cannot exist.

□

3.2.3.2 Min-cut based Timing Monitor Synthesis Strategy

The TDG effectively turns the interference problem into a reachability problem, where lower confidence "flows" along the dependence edges in $\mathcal{TDG}$ towards other nodes. A monitor is able to cut such a flow, as it alters the result of the confidence function for the particular node(s) it enforces. In order to configure a networked system such that the number of monitored parameters is small, a min-cut of the dependency graph is necessary. Applying a minimum-cut to a TDG, however, has three prerequisites: First, the flows that actually taint a confidence requirement need to be identified. Second, in these flows, the graph must be reduced to nodes of types that can be monitored, as for instance for a maximum busy-window node w^+ no monitoring and enforcement techniques are known. This reduction step, however, must preserve the structure of flow relations. Third, a capacity formulation for the edges of the reduced graph is necessary.

We address the problem by an iterative approach, addressing confidence requirement violations in descending order of the confidence requirement level. For each confidence requirement level i in the system, we collect all constraint nodes with a confidence requirement mismatch in $\mathcal{M}_i$:

$$\mathcal{M}_i = \{v_a : \Gamma(v_a) < \Gamma_{req}(v_a) = i\} \text{ with } v_a \in \mathcal{V} \tag{3.2}$$

By construction, all paths that can influence the confidence of the constraints in $\mathcal{M}_i$ are contained in the reachability graphs $\mathcal{TDG}^{reach}(c_j)$ of the mismatching constraints $c_j \in \mathcal{M}_i$. To construct the flow network we merge these reachability graphs into a single one $\mathcal{TDG}_i^{reach}$ and remove all nodes where the interference is bounded w.r.t. the confidence level i. I.e. only nodes with a confidence less than level i remain in the subgraph $\mathcal{TDG}_{i,red}^{reach} \subseteq \mathcal{TDG}$.

However, $\mathcal{TDG}_{i,red}^{reach}$ still contains nodes for which no monitors and enforcement techniques are known. We therefore eliminate nodes for which no monitor is known from the graph, while preserving all dependence relations (edge relations) from $\mathcal{TDG}_i^{reach}$ by adding appropriate edges. This is achieved

by Algorithm 4. It takes $\mathcal{TDG}^{reach}_{i,red}$ as well as a list of nodes for which no monitor type is known as an input. Node types for which no monitors are known are e.g. the nodes representing the multiple-event processing time (cf. Definition 2.3.1) for different schedulers, such as SPP (cf. Lemma 2.4.11) and SPNP (cf. Lemma 2.4.15). These nodes can be found by trivial iteration over $\mathcal{TDG}^{reach}_{i,red}$. In essence, Algorithm 4 links all predecessors of a removed node to its successors preserving the transitive dependence relations (cf. Lines 5 to 7). I.e. if a node is removed from the graph, all predecessors of the removed node are linked to its successors preserving the transitive dependence relations. The (again) reduced graph is returned by Algorithm 4 as $\mathcal{TDG}^{reach}_{i,mon}$.

Algorithm 4: reduce_to_monitorable($\mathcal{TDG}^{reach}_{i,red}$, non_monitorable_nodes)

Data:
non_monitorable_nodes : the nodes for which no monitor type is available
$\mathcal{TDG}^{reach}_{i,red}$: The reachability graph for confidence level i

```
1  TDG^reach_{i,mon} = TDG^reach_{i,red} ;
2  for n in non_monitorable_nodes do
3  |  predecessors = TDG^reach_{i,mon}.predecessors(n) ;
4  |  successors = TDG^reach_{i,mon}.successors(n) ;
5  |  for p in predecessors do
6  |  |  for s in successors do
7  |  |  |  TDG^reach_{i,mon}.add_edge(p,s) ;
8  |  |  end
9  |  end
10 |  TDG^reach_{i,mon}.remove_node(n) ;
   |  ;     /* implies the removal of all edges from and to node n */
11 end
12 return TDG^reach_{i,mon}
```

For illustration, we consider the CLM from Fig. 3.3. It models a system with a CAN bus and an ECU, that is reduced to the task graph and resource graph. In this example, the WCRT R_5^+ of τ_5 on the ECU Res_{ECU} has a confidence requirement level of 1 all remaining tasks have no confidence requirements, i.e. 0. The confidence values of input parameters can be found in Table 3.1. They are chosen such that the event models of τ_2 and τ_1 transitively interfere with R_5^+, i.e. $\mathcal{M}_1$ contains only the TDG node of R_5^+. The resulting $\mathcal{TDG}^{reach}_{i,red}$ is shown on the left-hand side of Fig. 3.5. Since monitors are only

known for event models and execution times (cf. Section 3.2.1), the set of non-monitorable nodes for Algorithm 4 is the complementary set of nodes representing event models and execution time bounds in the TDG. The result of Algorithm 4 is shown in the middle column of Fig. 3.5.

Table 3.1: Confidence values for specified parameters. Note that τ_4 is event dependent, and hence no confidence for the event model is specified.

Task	**confidence** C^+	**confidence** C^-	**confidence** δ_{in}^-	**confidence** δ_{in}^+
τ_1	1	1	0	0
τ_2	1	1	0	0
τ_3	1	1	0	0
τ_4	1	1	X	X
τ_5	1	1	1	1

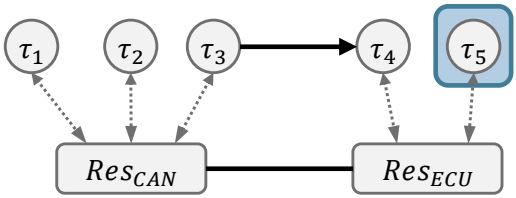


Figure 3.3: Example CLM to illustrate the difference between the min-cut and the greedy strategy. The blue-boxed task has a confidence requirement on its WCRT.

The last step to perform a minimum s-t-cut that separates the confidence requirement nodes $c_j \in \mathcal{M}_i$ from the interfering nodes (strict sense), is a suitable capacity formulation for the edges of $\mathcal{TDG}_{i,mon}^{reach}$.

Our desired result from the s-t cut of $\mathcal{TDG}_{i,mon}^{reach}$ is minimum cardinality set of nodes that require a monitor. Due to the duality of min-cut and max-flow problems, a suitable flow formulation, i.e. capacity assignment to the edges, is necessary. Since a monitor on a node turns all outgoing interference to bounded interference and not just for a specific dependency edge, we amortize placing a monitor into a specific node by setting the capacity of

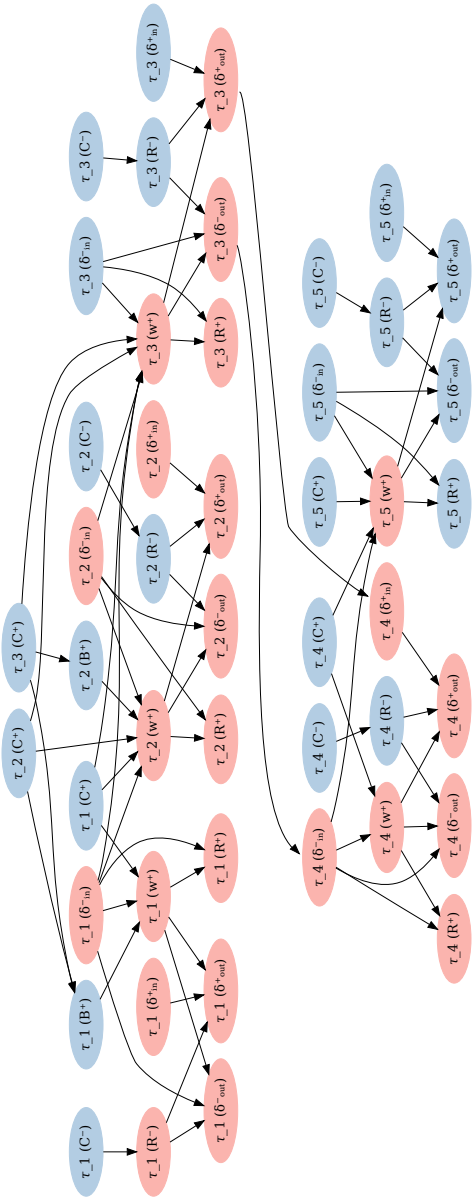
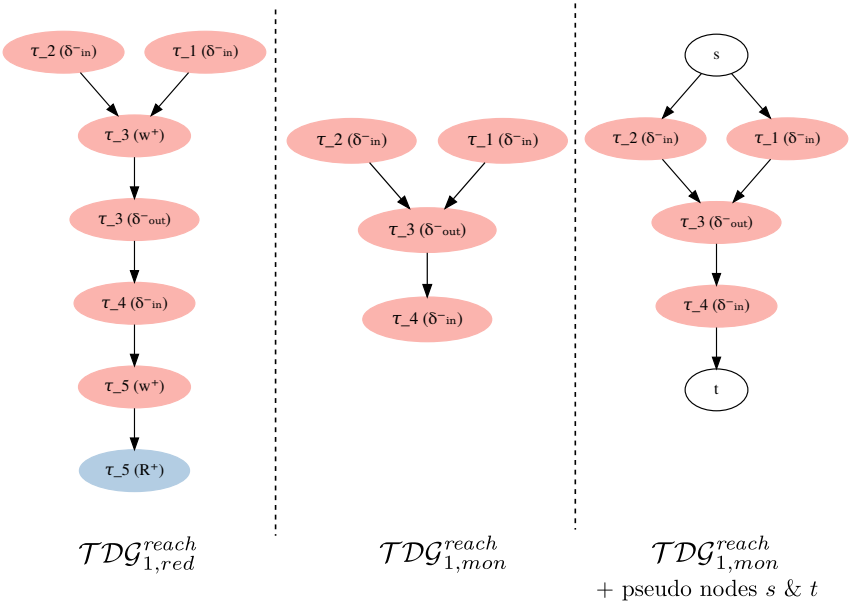


Figure 3.4: TDG of the min-cut example from Fig. 3.3. Blue indicates a confidence level of 1, red a confidence level of 0.



$\mathcal{TDG}^{reach}_{1,red}$ $\mathcal{TDG}^{reach}_{1,mon}$ $\mathcal{TDG}^{reach}_{1,mon}$ + pseudo nodes s & t

Figure 3.5: $\mathcal{TDG}^{reach}_{1,red}$ (left), the result of Algorithm 4 $\mathcal{TDG}^{reach}_{1,mon}$ (middle), and $\mathcal{TDG}^{reach}_{1,mon}$ with already inserted pseudo nodes s and t (right)

each outgoing edge to the node's out degree. Due to the property of flow preservation, i.e. the inbound flow to a node is equal to the outbound flow, this guarantees that a maximal amount of flow can be outbound and that the inbound capacity is the limit. Again, a maximum flow is dual to the minimum s-t-cut of the network.

In order to apply e.g. the Edmonds-Karp algorithm [EK72], to determine the minimum cut of the network a definition of the source s and sink t of the flow is necessary. $\mathcal{TDG}^{reach}_{i,mon}$ by its construction has at least one leaf that is a constraint with a violated confidence requirement. In order to define a single sink for the flow network we add a pseudo node t to the flow network and connect all the leaf nodes (out degree of 0) of $\mathcal{TDG}^{reach}_{i,mon}$ with t. Using the leaf nodes is necessary, as the original constraint nodes $c_j \in \mathcal{M}_i$ have been removed by Algorithm 4, i.e. in the step from $\mathcal{TDG}^{reach}_{i,red}$ to $\mathcal{TDG}^{reach}_{i,mon}$ (which contains only nodes for which monitors are available but preserves all the interference paths from $\mathcal{TDG}^{reach}_{i,red}$). Since in $\mathcal{TDG}^{reach}_{i,mon}$ the nodes with an out degree of 0 have been (transitively) connected to constraint nodes $c_j \in \mathcal{M}_i$ in $\mathcal{TDG}^{reach}_{i,red}$ this preserves the properties. To prevent that edges from the former leaf nodes to t become the dominating edges for the maximum flow, i.e. restrict the augmenting paths, their capacity is set to infinity. The input parameters are treated similarly. The pseudo node s is added and connected with all remaining input parameters in $\mathcal{TDG}^{reach}_{i,mon}$ and the capacity of these edges is also set to infinity. For the example system from Fig. 3.3 this is shown in Fig. 3.5 in the right column.

Based on the maximum-flow computation from [EK72] the minimum node cut set can be determined. The nodes in this set – if removed – would sever all paths from the input parameter nodes to the constraint nodes in $\mathcal{TDG}^{reach}_{i,mon}$. Consequently, monitors are added only for these nodes, resulting in a minimum number of monitors. These monitors then prevent that a node of lesser confidence can taint the confidence requirement of a timing constraint. As the flow network is constructed such that no other node of lesser confidence (strict interference) in $\mathcal{TDG}$ can reach any constraint in $\mathcal{M}_i$ the confidence requirement of all constraints in $\mathcal{M}_i$ are fulfilled for level i after inserting the monitors. Applied to the example system from Fig. 3.3, this results in a monitor for τ_4's input event model, more precisely its δ^- function. For comparison, the greedy strategy would synthesise monitors for τ_1 and τ_2's δ^- functions, as monitors are only placed on specified input parameters by the greedy strategy.

3.2.4 Experiments

To evaluate the effectiveness of both placement strategies we conduct a series of experiments. The structure of the synthetically generated cause-effect chains is derived from an automotive use case from a real-world research vehicle [Ber15].

3.2.4.1 Experimental Setup

The network of the platform consists of four Ethernet switches that are connected in a ring topology. Detailed insight into modeling Ethernet networks based on a task and resource model can e.g. be found in [TSAE16c]. In principle, a switch is modeled as a set of communication resources, where each communication resource models one output port. In consequence, the incoming links to a switch are connected to each output port where frames are queued. The arbitration and hence timing interference takes place there. Each switch is connected to two execution resources as ECUs, i.e. eight in total. The platform is depicted in Fig. 3.6. The ECU are scheduled under a

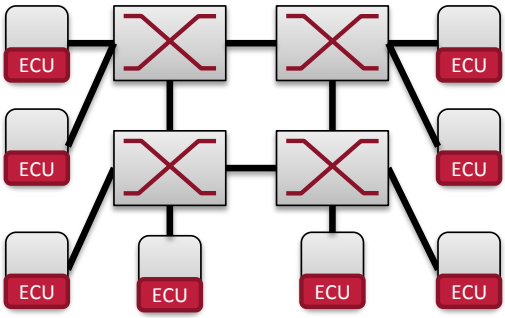


Figure 3.6: The platform to which cause-effect chains of specific length are randomly mapped to for the experiments, consisting of four switches connected in a ring topology with two ECUs connected to each switch

For simplicity and intelligibility each software block on a resource is mapped to an individual task. Communication of data dependent tasks on the same execution resource is assumed to be sampling based. The execution resource of the first task of a chain is chosen randomly based on uniform randomness. The execution resources of subsequent tasks are chosen according to a random distribution that favors higher distances between the execution resources. I.e. the higher the distance, the higher the probability of selecting a resource. The distance is measured in numbers of hops in the Ethernet network, whereas a hop is each arbitration of a message at an output port. Note that there is a probability, that the subsequent task resides on the same resource as its predecessor.

In the highly likely cases where two software blocks/tasks with data dependencies are mapped to different resources, tasks that handle the communication are generated. We therefore insert a communication task that models the communication stack on the sending and receiving ECU as depicted in Fig. 2.3 for chains with two times sampling (top right). This is done for every communication relation. The Ethernet frame sent over the network is modeled as a task on each output port (switch and resources) with event precedence between tasks. As the route for the frame, the shortest path between sending and receiving ECU is selected. The application tasks as well as the comm-stack tasks are triggered periodically by the system timer. We perform sampling based label exchange between the communication stack and the application task hosting the software block at both ends. Consequently, we assume data-age constraints between the software block in the execution task and the communication task and vice versa on the other end of the communication. To complete the overall end-to-end delay for the initial application chain we add response time constraints along the task chain that carries the communication.

Note that except for the assumption that we assign every software block to an individual task – which is only done for simplicity of the presentation – the setup resembles the practices in the automotive domain. Therefore, intra resource communication is performed sampling based and inter-resource communication is handled by a communication stack (cf. [AUT19b]).

To perform timing dependence and confidence analysis it is necessary that priorities and confidence values for the execution times and the initial event models are assigned. Furthermore, for each constraint of an application chain a confidence requirement is necessary. We refer to different assignments of initial confidences, confidence requirements and priorities as a variant of an application mapping. The confidence into the WCET of an

Ethernet frame, i.e. the frame size, is constant for a stream. Similarly, an Ethernet stream has the same priority on all Ethernet resources. Event Models generated by a common timer receive the same confidence, i.e. for all tasks on a resource the event model confidence is identical. The remaining parameters are chosen uniformly random. For monitor placement we assume WCET / BCET monitors (e.g. [HCBK12, NAME13]) as well as event model monitors (e.g. [HCBK12, NMA+12]). Consequently, the only TDG nodes for which no monitor can be placed are for R^+ / R^- and w^+.

3.2.4.2 Results and Interpretation

One experiment analyses one variant with both strategies. Experiments are conducted with an ascending number of application chains mapped to the platform. For each number of chains 10 different variants are generated. Furthermore, to investigate the impact of the mixture of *mixed-critical* and hence *mixed-confidence requirements*, two different distributions of confidence requirements are investigated. This is achieved by selecting different randomness distributions for the confidence requirement assignment in variant generation. The first distribution is a uniform distribution, i.e. confidence requirements between 0 and 4 are equally likely. The second distribution favors lower confidence requirements and is shown in Table 3.2. The rationale behind the second distribution is that realistic workloads only have a minor proportion of software that is actually highly critical and a large proportion that is mainly quality managed.

Table 3.2: Non-uniform distribution of confidence requirements

Confidence Requirement Level	**0**	**1**	**2**	**3**	**4**
Proportion	50%	20%	10%	10%	10%

We first turn our view to the results of the uniform distribution. In Fig. 3.7 the absolute number of generated monitors is shown as a boxplot, that depicts the median, the quartiles, and whiskers to show the rest of the distribution. Outliers are marked by diamonds. From the results in Fig. 3.7 we can observe that for a smaller number of chains, the min-cut strategy has an advantage, since in the median of synthesised monitors for less than 14 chains is always lower. The same holds for the maximum number of monitors required to achieve bounded interference. However, the min-cut is not necessarily to be

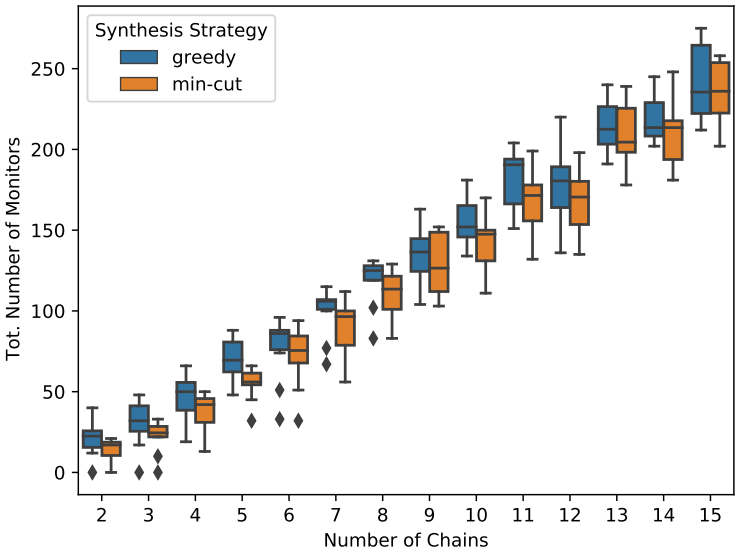


Figure 3.7: Number of monitors for different numbers of cause-effect chains assuming a uniform distribution of confidence requirements between 0 and 4 among the chains

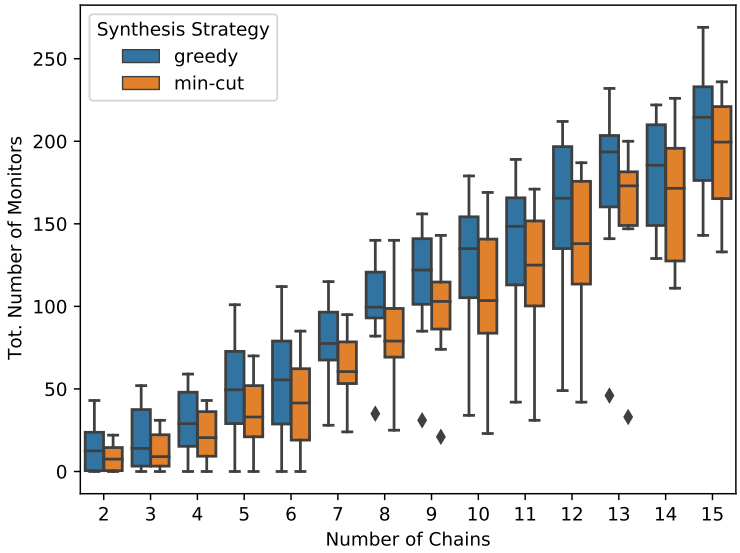


Figure 3.8: Number of monitors for different numbers of cause-effect chains assuming a that 50% of the CE chains have the lowest confidence requirement (confidence 0), 20 percent on confidence 1, and 10% on each confidence level form 2 to 4

favored since it has to be run per confidence requirement level that has to be separated from lower levels. The greedy strategy on the other hand places monitors on all input parameters that cause a confidence violation in one run. Consequently, if only few high confidence requirements are present this can lead to sub-optimal monitor placement. For instance if the min-cut for the high confidence requirement generates a placement "inside" the TDG and the min-cuts for next lower levels generate a placement similar to the greedy strategy where only input parameters are monitored. This can be observed in Fig. 3.9, for the uniform distribution of confidence requirements. The deterioration of the average improvement with an increasing number of chains is clearly visible in Fig. 3.9 that shows the relative improvement of the min-cut over the greedy strategy. For 8 chains and above a clear trend becomes evident that the median improvement stagnates around 6% to 8%.

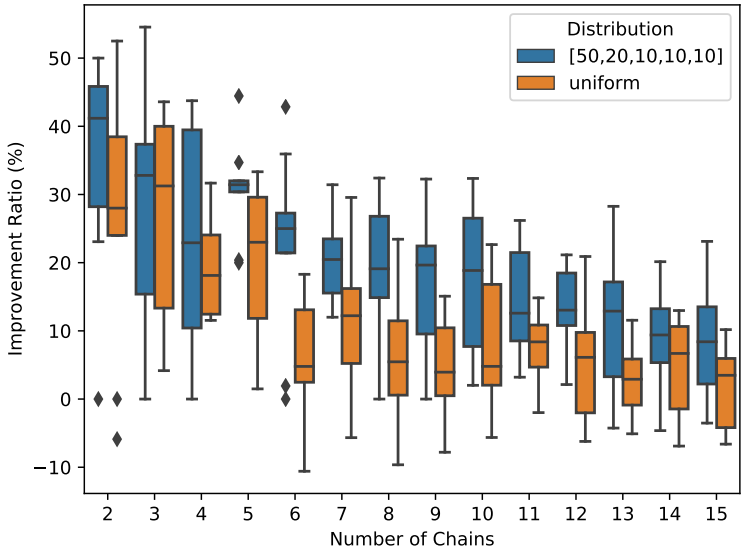


Figure 3.9: Relative Improvement of the min-cut strategy over the greedy strategy for the two distributions of confidence requirements

We attribute this deterioration to the fact that with an increasing number of cause-effect chains also the communication increases. This requires significantly more isolation of the communication as well. Our conclusion

is based on the fact that the majority of monitors generated with increasing number of chains, monitors output event models of communication tasks. I.e. the rate and pattern at which traffic can be injected into the network. Moreover, the communication drastically increases with more chains. The random process determining the placement of an execution task has a higher likelihood (non-uniform distribution) that tasks with a data dependency are mapped to most distant execution resource. Another point supporting this is that for an increasing number of chains the median ratio of monitored nodes vs. the total number of monitorable TDG nodes in the variant stays fairly constant (cf. Fig. 3.10), plateauing at 20% to 23% for 9 chains and above. We also conclude that consolidating the communication tasks into a single one per resource could further reduce the number of necessary monitors. Especially if this communication task monitors by design the rate and pattern at which frames are injected into the network at a high confidence. This could considerably contribute to a safe design w.r.t. safety that is based on timing requirements.

Fig. 3.10 highlights how systematically respecting end-to-end requirements improves the state-of-the-art, where all monitorable nodes would require enforcement. The figure shows that both strategies can reduce the amount of monitors compared to the state-of-the-art by more than 75%.

For the non-uniform distribution of confidence requirements from Table 3.2 we can observe in the results depicted in Fig. 3.8 that in total a smaller number of monitors is necessary. This is to be expected as 50% of the chains do not require isolation, i.e. bounded dependence, among them. This further explains the extreme cases where no monitors needed to be synthesised at all for 6 or fewer chains. However, the same trend, i.e. the min-cut strategy requires fewer monitors, as for the uniform distribution can be observed w.r.t. the median number of synthesised monitors. In fact, the improvement ratio, i.e. the improvement in % of monitors synthesised by the min-cut compared to the greedy strategy, is on average higher than for the uniform distribution.

Evaluating the monitoring overhead further, is highly implementation dependent. I.e. the cost associated with a monitor depends on the hardware platform, OS and run time environment. While implementations in a small RTOS can be done quite efficient w.r.t. code and memory usage [NMA+12], implementations for microkernels can be much more challenging. What really limits widespread use of monitoring is that most implementations ([NAME13],[NMA+12],[HCBK12]) require private hardware timers. Yet the number of timers is often limited to 2 up to 5 per ECU – depending on

concrete design – and the OSes typically need at least one. However, with a 75% reduction of monitors the number of required timers becomes realistic. E.g. the industrial case study (engine management system) from the 2016 WATERS challenge contains a maximum of 8 tasks per core. Consequently, a system-level monitoring scheme based on recent hardware platforms becomes feasible.

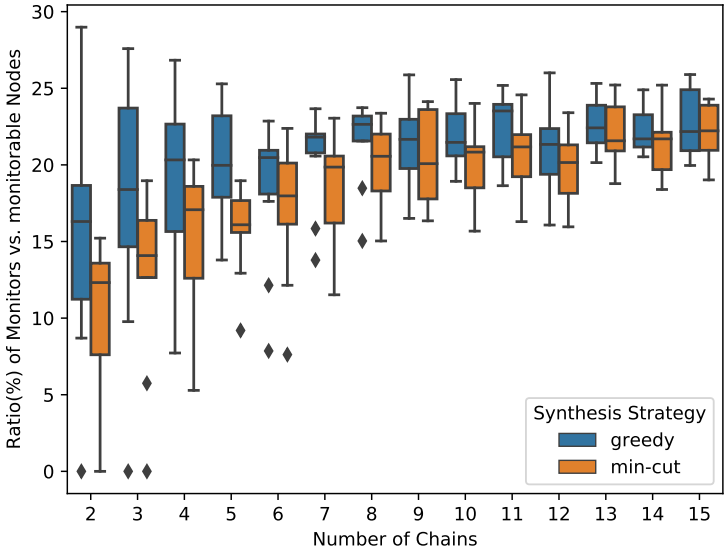


Figure 3.10: Ratio of synthesised monitors over monitorable TDG nodes

3.3 Summary

This chapter has approached the synthesis challenge for mixed-critical systems from two perspectives. The first one is the requirements engineering perspective that is intended to tackle issues raised by [Nan12], where the authors claim that a majority of safety-related serious consequences can be avoided by better requirements engineering. The requirements engineering method presented in Section 3.1 synthesises confidence requirements

on timing parameters in the BET model such as execution time bounds (C^+/C^-) or event models (δ^-/δ^+). It derives them with the help of the TDG data structure from safety requirements on timing analysis results such as WCRTs or maximum data-age, available through the cross-layer specification.

The second approach presented in this chapter builds on these assigned confidence requirements that would ensure a *sufficient* isolation in the sense that mutual interference is unlikely enough. The monitors instantiated in this process, however, require a meaningful configuration. There are two basic approaches how to handle the monitoring configuration:

The first one is to configure the monitors, exactly with the result from the CPA / modular performance analysis (MPA) timing analysis. The values obtained from there directly enforce the behavior observable in the model-based analysis. This behavior is, according to the analysis, a still tolerable one and hence correct behavior. However, it does not take into account that minimal design faults may exist, e.g. through the abstraction in the models. It might lead to premature termination of a task or deferring an activation unnecessarily. The resulting question with respect to the extra-functional timing requirements is rather, when is the intervention by the monitoring actually necessary? This leads to the second approach to configure monitors based on sensitivity analysis of the system. Based on the assumption that the extra-functional requirements are exhaustively described for all tasks in the system by a deadline, a number of approaches have been reported in the literature for real-time systems. In case of finding monitoring bounds, sensitivity analysis varies parameters of a system in order to find extreme points at which requirements are violated. Already [PDB97] reported on how the sensitivity bound of the worst-case execution time of a single task in a system with strictly periodic workloads can be found. But the approach assumes that only one task at a time would be allowed to exceed a parameter. Hence, it was later succeeded by [BNB07] where the authors also assume a strictly periodic task model, but present a solution in which a pareto front for variations in either the execution-time or the periods of all tasks in the system is computed. What make the problem challenging for CPA / MPA methods are discontinuities in the response-time function. Therefore, [RJE05] and [HJRE06] proposed a framework built around CPA that leverages binary search and genetic algorithms to compute the pareto front. In contrast to [BNB07], the authors in [RJE05] [HJRE06] are able to vary any kind of timing parameter for each pareto front, i.e. execution time bounds, period and jitter. [WTVL06] presents a similar approach for MPA.

However, all of these approaches require the knowledge of deadlines for each and every task in the system. While this assumption is reasonable for the tasks with critical timing, not all tasks in a system necessarily have an explicit deadline. In many cases, it is implicitly assumed that the deadline equals the period. Nevertheless, tasks with a low timing criticality that run as best-effort tasks often do not have an explicit deadline formulated as part of their extra-functional requirements from the function implementer. An open challenge which, among others, shall be addressed in the course of the next chapter.

Chapter 4

Predictable Design on Complex Platforms

The advent of higher-level driving functions according to the SAE taxonomy [Int18a] challenges development for E/E-Systems in two very particular ways. First, for conventional non-automated vehicles with *assistance* systems only (SAE Level 2), traditional safety concepts only require a safe state to be reached. Hence, timing errors "only" trigger a transition into the safe state, where the service is discontinued but no hazard is created. This is commonly referred to as *fail-safe behavior*. Behavior like this can e.g. be observed from lane-keeping assistance systems, where the system might even suddenly disconnect and the driver has to take over immediately. However, for performing the dynamic driving task in SAE-Level 3+ systems such a development approach for safety is impossible, as continued service is required. In contrast to the fail-safe behavior, a *fail-operational* behavior of the system is ultimately necessary for these systems. While it was sufficient to detect a fault in fail-safe architectures and contain the effects such that no hazard is caused by it, fail-operational either requires to completely mask the fault or continue operation in emergency operation before ultimately bringing the system into a safe state [Int18b, Part 1, Figure 5]. For SAE Level 3+ the emergency operation time can become extensive compared to the execution frequency of control software, e.g. if the vehicle shall be brought to a safe stop. A safe stop might potentially include driving for a few kilometers, e.g. to exit a highway/autobahn.

Hence, timing requirements inevitably become safety relevant and can not be avoided by a fail-safe design for timing faults any more. Note that monitoring and enforcement as presented in Chapter 3 is able to limit interference caused by a timing fault, e.g. exceeding the execution time assumption or the assumed event model. In consequence, it is able to prevent (cascading) faults of the tasks that would be interfered with. However, monitoring and enforcement can not recover the initially faulty job causing the interference. E.g. if a job is terminated after exceeding its WCET it will not produce output data. Consequently, fail-operational capabilities for such timing faults is the next open challenge.

A further challenge is that higher-level automation functions dramatically increase the need for computational power as software becomes ever more complex. Especially for highly integrated systems with a high behavior complexity as well as a high architecture complexity predictable timing behavior is important to be able to design effect chains composable. Composability is the property that the output behavior of one component is a compatible input behavior for another. Particularly for timing this is challenging, as the timing of e.g. one software component can not be assessed in isolation. How a design can be achieved where timing is predictable, the safety assured, and the safety properties are traceable such that a high degree of composability can be achieved is the challenge, nevertheless it is a sub-challenge of the second challenge.

In the following section, intermediate conclusions for the design and integration of such systems are drawn, taking into account the methods presented in Chapter 3. Based on these conclusions the implications on the research hypothesis are studied. A possible solution to the arising questions, based on the logical execution time (LET) paradigm is presented further on. SL-LET will be introduced as an abstraction of the BET model, introduced in Section 2.1 and embraced for timing dependency analysis so far. This solution also maintains the traceability of safety requirements across model-layers as introduced before.

4.1 Intermediate Conclusions

Let us first consider the second challenge of increasing design and architecture complexity. Systems for SAE levels 3+ have high computational requirements, as they e.g. have to process a 360 degree input for perceiving and interpreting their environment. This is substantially more than

the state-of-the-art ADAS. For systematic redundancy, inputs of different sensor types must be fused into a joint environment model. Hence, data input inevitably has to be transported over in-vehicle networks and hence the functions become distributed. While networks in isolation, i.e. without considering execution and computation on end stations, can behave quite deterministic. The consideration of ECUs which fulfill the demand for more computational power make the system less and less predictable w.r.t. its timing behavior. The variability of execution times grows with architectural parameters, e.g. the cache-miss penalty, the costs for pipeline stalls and for control-flow mispredictions [Wil18]. Particularly, the spread between the observed average execution times and true conservative WCETs estimates complicate the situation. Recently, also the trend towards consolidating multiple ECUs into one ECU with computationally powerful multi-core processors adds additional complexity. The trend for consolidation requires virtualization techniques as they are nowadays state-of-the-art in cloud computing and internet server-rental. While basically all the virtualization techniques separate the virtual machines in terms of interference in their data, the timing can substantially change when software is migrated into virtualized systems. Further microarchitectural mechanisms like translation lookaside buffers (TLBs), branch prediction, and paging together with memory management units (MMUs) are no recent developments, but they have hindered the estimation of tight and truly conservative WCETs for more than two decades now [WGR+09]. The challenge is that the common case – for which the mechanisms are optimized – is often faster than the worst-case by a considerable factor and hence the mechanisms lose their advantage for hard real-time designs. For some mechanisms the decision whether the worst-case can actually be observed, can not be answered by rigorously analysing the software in isolation. As an example, one can consider caches. The hit-and-miss behavior is not only dominated by the software currently using the cache, but rather by the software running before and after running a particular job, as well as the software running during preemption of that job. Cache management techniques can help to improve bounds on the timing behavior. With this goal and the intent to isolate tasks in multicore environments, cache management mechanisms for real-time systems have been studied since the introduction of caches. [GAM+15] provides an overview of hardware and software mechanisms studied in the literature from 1990 to 2014. The authors of [GAM+15] conclude that: "[…] a certain level of hardware support is necessary to provide strong isolation guarantees to concurrently executing tasks[…]. Even when software-only implementation is possible, such solutions are typically more cumbersome, more difficult to

certify and/or add more overhead compared to hardware solutions.". Until today, such techniques have not made the necessary impact in closing the spread between average observed execution time and WCET bounds.

This wide spread of possible core execution times (CETs) becomes a design problem, when it comes to integration of real-time software. On the one hand, the software under consideration typically executes around the median CET, only in very rare cases extremely higher or lower CETs are observed. On the other hand, a large gap between conservatively estimated worst-case and actually possible worst-case in a given configuration has an impact on verification of timing requirements. The management perspective, which favors high resource utilization to save costs, will argue that the "free" utilization between an over provisioned median CET and the overly conservative worst-case estimate can be used for additional software. However, only using conservatively estimated WCETs for CPA or MPA to verify latency requirements, will give truly conservative results (assuming the used event models/arrival curves are similarly conservative estimates) [HHJ+05][TCN00]. However, this will also result in higher values for computed response times, which can easily result in violated latency requirements.

Obviously, one can argue that in the case where the latency requirements of the system does not hold when assuming conservative estimates for the CETs, the system must not be built. However, in practical testing such systems basically do not fail, as the probability of the CETs assuming the value of the conservative estimates all at once is almost negligible. A similar situation arises for the formal bounding of event models. For variations in the event model typical worst-case analysis (TWCA) [QBH+14] can be applied to bound the effect. Similarly to variations in the event model, the approach of TWCA (e.g. [HQE14],[Ham19],[XHK+15],[HEQ+17]) could be used to encode the additional execution time (i.e. the time between typical case and conservative worst case) as overload. However, the limitation to this is also the accuracy to which the *typical* and *overload* behavior is known. Hence, the estimate suffers from the same shortcoming as conventional CPA when a non-conservative estimate on the WCET is used.

However, a quite extensive number of research in the fields of CPA and MPA seems to have adopted the view that tight conservative WCETs exist [Wil18].

This is despite the fact, that even current research on contemporary processor architectures can not decrease the gap between "worst-case testing" and true conservative estimates. In fact, the biggest influence factor for multicore architectures – the shared cache – increases even if recent WCET timing analysis techniques are used [LGR+16]. A problem that is known

for over a decade now [WEE+08]. Even modern hybrid approaches where measurements are combined with static code and microarchitecture analysis are unable to solve the problem to a degree where it would be accepted in the industrial state of the art [KPWF19]. In conclusion, the design problem of providing conservative timing guarantees to verify timing constraints remains, if only static WCET analysis in combination with CPA or MPA is used. This is due to the fact that CPA and MPA are unable to produce conservative bounds if non-conservative WCET bounds are used for the computations. Chapter 3 has shown a loophole out of this problem by enforcing model behavior at run-time where e.g. WCETs are not conservatively known. Synthesising monitors into a design, where designers are unsure about the correctness of a WCET bound, tries to engineer around this problem. However, this development flow has a disadvantage w.r.t. composability of effect chains. Composability in this context is the property that certain properties of a system only depend on properties of a module. Composability is violated if individual properties of one or more modules depend on the composition of the systems [SSE05] [PS08].

The flow fails to provide early answers to two significant questions that arise during design. First, the end-to-end latency of an effect chain (or subchain) can not be easily determined, since this requires to spawn a complete analysis in the BET model. Although the analysis could be part of a tool supported continuous integration flow, the data necessary to parameterise it is typically not available early in the design. Second, for the end-to-end "age" of data at a consumer in an effect chain only upper and lower bounds can be computed in the BET model. With the increasing gap between best case and worst case, as well as the gap between average case and worst case, the spread between the lower bound and the upper bound of the data age becomes larger. Furthermore, the jitter between upper and lower bound on the data age remains unknown, making it difficult to analyze the impact on the function. If we for instance consider the lateral controller from Fig. 1.5, the actual control quality depends on the jitter behavior of the data age.

The controller example, is just one representative of applications where the data and the data's timing is crucial. Particularly in the ADAS domain and the conditional and highly automated domain, data-centric applications are prevalent. These applications have in common, that not only their execution behavior is of high interest but the timing of data that they process and produce. Similarly, their execution time depend on the semantics of the data they process.

Conclusions:

- A Data centric specification is necessary that describes the system's properties based on response-times rather than necessary execution times.
- Specify expected timing of data-flow and provide an implementation strategy that ensures this.

Extensions to the Research Questions:

Question 4: How can data-centric effect chains' timing safety requirements be specified, verified and validated?

Question 5: Is there another solution to the problem of needing ever more engineering effort to enable timing-safe fail-operational systems?

4.2 SL-LET as a new Design Paradigm

In the cross-layer model introduced in Section 2.1 the BET timing model serves the dual purpose of implementation model and design model. An implementation model in this context is a model that describes the run-time (timing) behavior, whereas the design model rather describes the properties desired by the designer. The BET model as introduced in Section 2.1 is built around the concept of events, i.e. it formally captures behavior of event streams (traces) and is well suited to compute corner-case behavior. For instance the longest time from an activation event of a task with an execution time between a best-case and a worst-case execution time estimate, i.e. the worst-case response time (WCRT). Yet, applying the BET model often requires to make a number of assumptions, such as that OS overhead is abstracted in the WCET estimates. A particular limit of the model is that data flow is not explicitly modeled in general. While data flow can be mapped to an (explicit) event flow in certain situations, like in the analysis of Ethernet or CAN networks, to compute data-ages and responses times of messages, there is no general way of describing data flow in the BET model. Particularly not if data is communicated sampling based, i.e. without explicit events in the sense of the BET model like the activation or termination of a task. However, as shown in Section 4.1, applications with increasing behavioral complexity, are often data-centric and require deterministic and predictable timing. A possible solution to the challenges formulated in Section 4.1 is to separate the design model from the implementation model.

The system-level logical execution time (SL-LET) concept as a design model allows applying methods to cope with both challenges formulated in Sec-

tion 4.1. To better understand the advantage of SL-LET the property of *timing composability* is introduced first:

Definition 4.2.1: Timing Composability

Composability is the property that (timing) requirements can be verifyed soleyly on the specification of an individual component and its interfaces.

This is a more stringent view on system design than the simpler property of *compositionally*:

Definition 4.2.2: Timing Compositionality

Compositionality is the property that (timing) requirements can be verfied based on (formal) rules from multiple component descriptions and their parameters.

Obviously it has to be clarified what can be regarded as a *component* as it is an abstract concept in the context of the CLM form Section 2.1. In the proposed CLM more than one interpretation for a component is possible. In the following a top-level view is adopted. From the top-level perspective, a component in the CLM is either an application that consists of a set of functions $F_{app} \subset F$ or a single function $f \in F$. As the focus here are timing requirements, we resolve to which tasks the function(s) map in the CLM, since the timing model represents the timing information. Hence, the (timing) interface of an abstract component is the timing parameters of all tasks belonging transitively to the component. In the example depicted in Fig. 4.1 two functions are depicted, f_a in orange and f_b in blue. The colors of the model elements of the software model as well as the timing model indicate that they (transitively) map to either f_a or f_b. Let us focus of on the (timing) requirements of f_b. The function is implemented by two software blocks, where b_{b1} produces data (L_{b1}) that is subsequently read by b_{b2}. Let us assume that this cause-effect chain has a data-age requirement d_b^{age} and as design and implementation model for timing the BET model is used. If the system in the example would exhibit composability, the requirement d_b^{age} would be verifiable based on the parameters of the blue model elements in Fig. 4.1 alone.

Yet, as we have seen in Section 2.5 the timing in this example not only depends on the timing parameters of all tasks on Res_2. This can be revealed

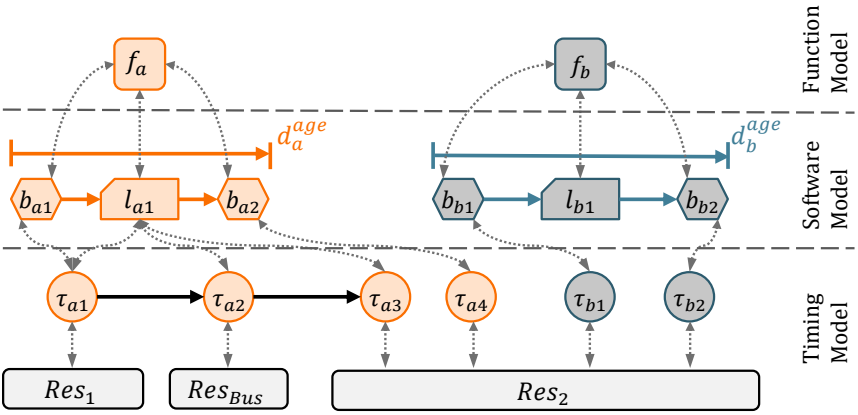


Figure 4.1: Example system with two functions, containing one cause-effect chain each

A consequence of the separation of design and implementation model is that it introduces another abstraction layer during system design. The abstraction helps to gain flexibility without having to consider certain implementation peculiarities already at design time. Obviously, this abstraction requires that correspondence rules with the implementation model must be existent. At the first glance, this looks like as if this step only introduces additional complexity in the design process. In fact, the effect is to the contrary. In the following, SL-LET will be introduced as the additional design model. To maintain the correspondence between SL-LET and the implementation, run-time mechanisms are necessary that ensure LETs [BE18]. What formal requirements are necessary for an implementation following BET execution semantics is shown in Section 4.5

4.2.1 The Design Principle of Logical Time

The idea behind logical execution time (LET) is that (physical) task execution is abstracted into logical time. First ideas for such abstract programming models date back to the *Giotto* language [HHK03]. The abstraction w.r.t. reading and writing data, and performing computations on it, is that LET

conceptually follows a strict read-execute-write behavior [KS12]. This divides a task in three phases: First, inputs are read in the read phase, second, processing/execution takes place, and the third phase writes the data such that it can be read by other tasks after the end of the phase. The time for all three phases combined is the LET. Fig. 4.2 depicts two jobs of a periodic LET task, recurring every P_i. The read phases and write phase in LET, however, are special and three assumptions are made: First the abstraction assumes, that reading and writing happen in zero-time, i.e. that no time is consumed for reading inputs and writing outputs. It reduces the read phase and write phase to particular events in time. Second, the read phase occurs instantaneously with the activation and output data is valid at the end of the LET. The third assumption is a consequence: Execution happens arbitrarily within the LET. No assumptions are made as to how the execution is performed, e.g. whether preemption occurs or that the execution starts directly after the read event. Note that this does not imply that a task has to execute until the write event, the idea is merely that outputs of a particular job are *published* to data-dependent tasks at the write event. It is, however, assumed that published data remains valid until new data is published at the next write event. Since publishing times, i.e. the write phase, as well as reading times are known by design, composable deterministic data flow can be designed. In the example in Fig. 4.2 any task consuming data from the depicted task is known to read the data from the n-th job starting at $n \cdot P_i$, if

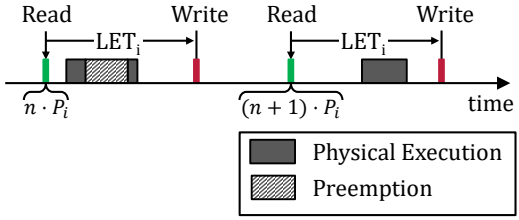


Figure 4.2: LET's Read-Execute-Write Concept for a periodic LET task activated every P

Obviously, an implementation must be able to mimic the assumed read and write behavior. Implementations like [BE18] have shown this for multicore

platforms with shared memory by reducing the read and write events to atomic pointer swaps, hence the *zero-communication time* assumption holds. As such, it has already found its way into the AUTOSAR Timing Extensions [AUT19a]. However, if communication over shared buses or networks such as Ethernet is used the communication latency must be taken into account. This is done by considering the communication as an LET task of its own, and is the extension of the plain LET concept from [HHK03], [KS12] referred to as SL-LET. SL-LET also takes into account that individual "plain LET islands" can experience a synchronization error of their clocks that coordinate the logical time. In the following this general idea of SL-LET is described formally on the basis of [EKG18].

4.2.2 The SL-LET System Model

Section 2.1 has introduced the BET model as a model to formally describe timing behavior of a system. As discussed in the previous section, we will now introduce the SL-LET model as an alternative design model. How an SL-LET design can be mapped to an implementation where the behavior corresponds to BET behavior semantics is later introduced in Section 4.3.

The definitions provided in this section are based on the initial work of [KS12] for plain LET and the system-level extensions for SL-LET in [EKG18]. Important deviations and extensions to these definitions are mentioned.

Definition 4.2.3: System Platform

The resource platform $\mathcal{P}$ is an extended version from Section 2.1. The platform $\mathcal{P} = \{\mathcal{R} \cup \mathcal{C} \cup \mathcal{M}, \mathcal{E}\}$ is a directed graph where the set of nodes consists of three subsets: $\mathcal{R}$ is the set of all *processing resources*, $\mathcal{C}$ the set of all *communication resources*, and $\mathcal{M}$ the set of all *memories*. The set of all edges $\mathcal{E}$ connects the individual platform elements.

LET tasks execute on processing resources and are defined as follows:

Definition 4.2.4: LET-Task

A *LET Task* λ_i is a recurring task with period P_i and offset O_i. It exhibits deterministic read and write behavior: For the j-th job $\lambda_{i,j}$ of the task, the inputs are read at $\hat{R}_{i,j} = (j-1) \cdot P_i + O_i$, and the outputs are written at $D^{min}_{i,j} = \hat{R}_{i,j} + LET_i$. The time between the read and the write event of data labels is referred to as the logical execution time LET_i of task λ_i. The output data interval in which output data of a job $\lambda_{i,j}$ is valid, is defined as $D_{i,j} = [D^{min}_{i,j}; D^{max}_{i,j}[$. Output data is valid until new data is produced by the next job, hence $D^{max}_{i,j} = \hat{R}_{i,j+1} + LET_i = D^{min}_{i,j+1}$.

Similarly to BET tasks, jobs are defined:

Definition 4.2.5: LET-Job

An instance of a LET task λ_i is referred to as a job, whereas the j-th job is denoted by $\lambda_{i,j}$. Two instances $\lambda_{i,j}$ and $\lambda_{i,k}$ with $j \neq k$ are independent in their execution unless data dependencies are explicitly modeled.

The Definition 4.2.4 is an extension to the one provided in [EKG18] which defined LET tasks based on [KS12] and [HHK03]. Particularly, the activation offset O is introduced here, which allows describing desired timing behavior of dataflow such that data from a producing task is read by a subscribing/consuming task immediately after publication. The offset parameter e.g. allows aligning tasks with identical periods *back-to-back*, i.e. that data produced by the first directly becomes available to the successor without delay. A similar concept was adopted by Martinez in [MSB18] and Becker in [BDM+17] for analyzing systems with plain LETs. However, no formal definition of LET tasks was provided and the LET was only used as a model mechanism to describe an implementation property. The contribution in this thesis is to use SL-LET as a *design model*, i.e. to formulate requirements towards the implementation (cf. Section 1.1 and [LS18]). While for plain LET tasks some publications implicitly assume that $LET_i \leq P_i$, this restriction does not literally appear in e.g. [KS12] or [HHK03]. Kirsch and Sengupta in their survey of programming paradigms [SK07], however, assume $LET_i = P_i$. Here, no restriction on the length of the LET is made, i.e. a LET greater or smaller than the period is explicitly allowed. A side effect of this defini-

tion is that it defines *when* the life-time $D_{i,j}$ of job's output data begins or ends, respectively. Nevertheless, the length of the data interval $|D_i|$ equals the period P_i. This follows trivially from the data interval's definition in Definition 4.2.4:

$$\forall\, j: \quad |D_{i,j}| = D_{i,j}^{max} - D_{i,j}^{min} = \tag{4.1}$$
$$= \hat{R}_{i,j+1} + LET_i - (\hat{R}_{i,j} + LET_i) = \tag{4.2}$$
$$= j \cdot P_i + O_i - (j-1)P_i - O_i = P_i \tag{4.3}$$

A notable extension of this definition, compared to others in the context of SL-LET is that it introduces a fixed but arbitrary zero-point in the time domain and allows to explicitly address jobs and data dependencies between jobs. This fact is an important property and revisited in the next section in Definition 4.3.2 and Definition 4.3.3.

Data input and output is explicitly modelled in SL-LET:

Definition 4.2.6: LET-Label

A data element being read or written by LET tasks is referred to as a data label $_i \in \mathcal{L}^{LET}$. Each label has a unique writing task, but can be read by multiple tasks. Further, each data label resides in a memory $m_x \in \mathcal{M}$.

Data labels, like LET tasks, also have instances:

Definition 4.2.7: LET-Label Instance

A data label instance $_{a,j}$ is the instance produced by the j-th job of its producing task, i.e. $\lambda_{i,j}$. A label instance is valid, i.e. can be read, only during the data interval of the job producing it, i.e. $D_{i,j}$. The label instances reside in the memory, the label is mapped to.

Defining this becomes necessary, since Definition 4.2.4, like later on Definition 4.2.9, allows arbitrary values for the LET. As pointed out before, previous definitions of LET and (SL-LET) did not reason on this fact, particularly not for the case of $LET_i > P_i$. An important consequence of the label concept introduced here is that, any state of a task must also be encoded in a label, if it shall be preserved across multiple jobs. Hidden state, i.e. data which is not modeled, is prohibited in this (SL-LET) model.

In the plain LET model, any number of LET tasks can communicate based on the zero-communication time assumption, where a write and read event can be instantaneous. All LET tasks for which this is possible execute in the same *time zone* and read and write their labels from memories in this time zone.

Definition 4.2.8: Time Zone

A *time zone* Z_a is a subgraph of the hardware platform. Within a time zone the (plain) LET programming model is valid. Particularly read and write operations to data labels in Z_a can be performed in a time interval $\Delta t = 0$.

To lift the limitations of the (plain) LET model, and to allow communication with latencies, LET interconnect tasks are introduced.

Definition 4.2.9: LET Interconnect Task

A *LET Interconnect Task* Φ_i is defined as a generalized LET Task that copies data labels from a memory in a source time zone $m_x \in Z_a$ to a memory location in a remote time zone $m_y \in Z_b$. It consumes service on a composed resource, namely on all communication and processing resources involved in the copying process between the time zones Z_a and Z_b.

Note that to account for longer or shorter transmission times, $LET_{\Phi,i}$ of Φ_i is not restricted in general, e.g. to P_i, and might be significantly longer. However, as shown in Eq. (4.3) the length of the data interval $|D_i|$ always equals the period.

Note that Λ^Φ denotes a set of interconnect tasks according to the above definition, while Λ^λ denotes a set of LET tasks according to Definition 4.2.4. An element $\lambda_i \in \Lambda$ from the union set $\Lambda = \Lambda^\lambda \cup \Lambda^\Phi$ can be either an interconnect or plain LET task. Extending [EN16], for both types of LET tasks, read and write sequences of jobs are defined:

Definition 4.2.10: LET Read Sequence

A *read sequence* of a task $\lambda_i \in \Lambda$ is a function

$$\sigma_i^{LET} : \mathbb{N}_0^+ \rightarrow \mathbb{R}_0^+$$

where $\sigma_i^{LET}(n) = \hat{R}_{i,n}$ denotes that the n-th job reads the input labels valid at time $\hat{R}_{i,n}$.

Definition 4.2.11: LET Write Sequence

A *write sequence* of a task $\lambda_i \in \Lambda$ is a function

$$\omega_i^{LET} : \mathbb{N}_0^+ \rightarrow \mathbb{R}_0^+$$

where $\omega_i^{LET}(n) = D_{i,n}^{min}$ denotes that the n-th job publishes its output labels at time $D_{i,n}^{min}$.

A particular aspect of an SL-LET description is that timing of data, i.e. the time when tasks perform reads and writes, but not the actual execution itself is time triggered. A consequence arising from that fact is that in each time zone, task triggering must depend on a common clock source in order to ensure synchronicity. While this can be achieved e.g. by a common timer for all cores of a multiprocessor setup, perfect synchronization of clocks between different ECUs is practically impossible. Hence, interconnect tasks not only bridge between "islands" in which the (plain) LET programming is valid, but also between different clock sources. To have a common understanding of time, a synchronization between time zones is necessary in SL-LET designs.

Definition 4.2.12: Bounded Synchronization Error

A synchronization mechanism creates a global time base among all time zones, whereas the local time is a local approximate of the global time base. The maximum difference between any two approximates of the global time must be bounded from above by a known limited error ϵ.

Interconnect tasks and (plain) LET tasks can be combined in a joint graph to express the deterministic data flow intended by SL-LET.

Definition 4.2.13: LET Task Graph

A *LET Task Graph* is a directed graph $\mathcal{LG} = (\Lambda \cup \mathcal{L}^{LET}, \xrightarrow{r} \cup \xrightarrow{w})$, where the set of vertices consits of the union set of all LET tasks and interconnect Tasks $\Lambda = \Lambda^{\lambda} \cup \Lambda^{\Phi}$ and the set of all data labels $\mathcal{L}^{LET}$; the set of edges is denoted by the union of the set all read access to data labels $\xrightarrow{r}$ and the set of all write access to data labels $\xrightarrow{w}$.

Three left-total relations map the nodes of the LET task graph to the nodes of the platform graph:

$$\rho_{\lambda}^{LET} \subseteq \Lambda^{\lambda} \times \mathcal{R} \tag{4.4}$$

$$\rho_{\Phi}^{LET} \subseteq \Lambda^{\Phi} \times \hat{\mathcal{R}} \qquad \text{with } \hat{\mathcal{R}} = \mathcal{R} \cup \mathcal{C} \tag{4.5}$$

$$\rho^{LET} \subseteq \mathcal{L}^{LET} \times \mathcal{M} \tag{4.6}$$

Eq. (4.4) and Eq. (4.6) are required to be right-unique, i.e. each task and label are mapped to exactly one resource or memory element respectively, hence they can also be interpreted as functions. The mappings of Eq. (4.5) are constrained such that for a fixed but arbitrary $\Phi_i \in \Lambda^{\Phi}$ the relation ρ_{Φ}^{LET} yields elements from $\hat{\mathcal{R}}$ such that there exists a walk in $\mathcal{P}$ that contains the elements from $\hat{\mathcal{R}}$.

4.3 SL-LET as a Design Model in the CLM

To incorporate SL-LET as an additional model in the CLM, the SL-LET model must be represented as a joint graph. Therefore, the platform graph, the LET Task Graph, and the resource mappings are combined into one graph $\mathcal{LM}$ to form the *SL-LET design model.*

Definition 4.3.1: SL-LET design model

The *SL-LET design model* is a graph $\mathcal{LM} = (V, E)$, where

- V is the union set of the vertex sets of platform and LET task graph, i.e.

$$V = \Lambda \cup \mathcal{R} \cup \mathcal{C} \cup \mathcal{M}$$

- and E is the union set of edges of the system platform, the LET task graph, and the edges defined by the resource mapping relations $\rho_{\lambda}^{LET}, \rho_{\Phi}^{LET}, \rho^{LET}$i, i.e.:

$$E = \overset{r}{\rightharpoonup} \cup \overset{w}{\rightharpoonup} \cup \rho_{\lambda}^{LET} \cup \rho_{\Phi}^{LET} \cup \rho^{LET}$$

To impose constraints on a design described by the SL-LET design model, cause-effect chains can be defined:

Definition 4.3.2: LET Cause-effect chains

A *cause-effect chain* $\mathcal{C} = (v_1, ..., v_n)$ is the vertex sequence of LET tasks and/or interconnect tasks $v_i \in \Lambda$ in $\mathcal{LG}$ of a finite directed walk in the LET graph, i.e. without the labels between tasks.

Note, that for notational simplicity cause-effect chains are in the following often written in the form $\mathcal{C} = (\lambda_1, ..., \lambda_n)$ with $n \in \mathbb{N}^+$ with the intention that $\lambda_i \in \Lambda$, i.e. λ_i can be either an interconnect task according to Definition 4.2.9 or a "proper" LET task according to Definition 4.2.4 – despite the fact, that the notation might suggest otherwise, i.e. λ is used for interconnect tasks as well. An instance of a cause-effect chain describes concrete data-flow, i.e. jobs of successors in $\mathcal{C}$ read labels produced by jobs of their predecessors. Since tasks in SL-LET can be executed with different periods and LETs, more than one instance of a chain $\mathcal{C}_i$ is possible. The exact number of instances is determined by the number of possible data-paths via LET task instances. Cause-effect chain instances are formally defined as:

Definition 4.3.3: LET Cause-effect chain instance

The j-th instance $\mathcal{C}_{i,j}$ of a cause-effect chain $\mathcal{C}_i = (\lambda_a, ..., \lambda_{a+k}, ..., \lambda_b)$ with $k \in \mathbb{N}^+$ is a data-flow ordered set of jobs of the tasks in $\mathcal{C}_i$. It is the result of the relation $\mathscr{C}$ from the ordered set $\{\lambda_{a,x} : x \in \mathbb{N}^+\}$ of jobs from λ_a and the ordered set $\{\lambda_{b,x} : x \in \mathbb{N}^+\}$ of jobs from λ_b:

$$\mathscr{C}_{a,b} : \{\lambda_{a,x} : x \in \mathbb{N}^+\} \rightarrow \{\lambda_{b,x} : x \in \mathbb{N}^+\}$$

that maps jobs $\lambda_{a,n} \in \{\lambda_{a,x} : x \in \mathbb{N}^+\}$ to elements from $\{\lambda_{b,x} : x \in \mathbb{N}^+\}$ if a data-flow from $\lambda_{a,n}$ to the elements in $\{\lambda_{b,x} : x \in \mathbb{N}^+\}$ is possible, based on the read event times and data intervals of the jobs from the tasks in $\mathcal{C}_i$.

Note that, $\{\lambda_{a,x} : x \in \mathbb{N}^+\}$ and $\{\lambda_{b,x} : x \in \mathbb{N}^+\}$ are the sets of possible jobs for λ_a and λ_b.

Theorem 4.3.4: Injectivity of LET cause-effect chain instances

The relation $\mathscr{C}_{a,b}$ is injective.

Proof 4.3.5:
First, consider two arbitrary but adjacent LET tasks $\lambda_c, \lambda_d \in \mathcal{C}_i$. Inorder for

$$\mathscr{C}_{c,d} : \{\lambda_{c,x} : x \in \mathbb{N}^+\} \rightarrow \{\lambda_{d,x} : x \in \mathbb{N}^+\} \tag{4.7}$$

to be injective, it must hold for every $\lambda_{d,m} \in \{\lambda_{d,x} : x \in \mathbb{N}^+\}$ that there exists *at most one* $\lambda_{c,l} \in \{\lambda_{c,x} : x \in \mathbb{N}^+\}$. This is the case, since:

i) data intervals of jobs of λ_c do not overlapp by construction (cf. Definition 4.2.4), i.e. $\nexists\, D_{c,p}, D_{c,q} : D_{c,p} \cap D_{c,q} \neq \emptyset$ for $p \neq q$

ii) read events of jobs of λ_d can consequently only be part of one data interval of λ_c's instances: $\exists!\, D_{c,q} : \hat{R}_{d,r} \in D_{c,q}$. In conseuence, there is a unique predecessor (which is an even stronger property than required) to any fixed but arbitrary $\lambda_{d,r}$.

> Second, since the composition of injections like Eq. (4.7) is again an injection, this follows for the entire cause-effect chain instance $\mathcal{C}_{i,j}$ based on the initial job $\lambda_{a,n}$.
>
> □

Theorem 4.3.4 and its proof is of importance, as it specifies an explicit, deterministic and predictable timing of data-flow, i.e. SL-LET can be used as an engineering model for the timing of data-flow. This is in contrast to cause-effect chains in the BET model, as it is impossible to decide a-priori which cause-effect chain instances are possible due to the lacking injectivity property (cp. Definition 2.1.13). As an example for an SL-LET cause-effect chain consider the chain from λ_a to λ_b depicted in Fig. 4.3, where λ_b oversamples the label produced by λ_a in a $1:2$ ratio. In this example, the output of $\lambda_{a,1}$ is read by two jobs $\lambda_{b,3}$ and $\lambda_{b,4}$, i.e. $\mathscr{C}(\lambda_{a,1}) = \{\lambda_{b,3}, \lambda_{b,4}\}$. Hence, two instances exist which originate from $\lambda_{a,1}$:

$$\mathcal{C}_{i,1} = (\lambda_{a,1}, \lambda_{b,3})$$
$$\mathcal{C}_{i,2} = (\lambda_{a,1}, \lambda_{b,4})$$

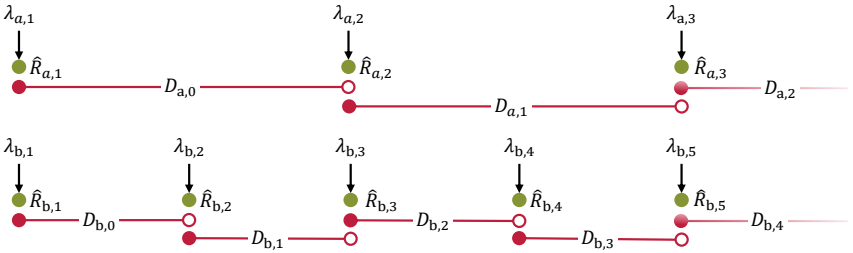


Figure 4.3: Dataintervals of an LET effect chain from λ_a to λ_b with oversampling

A consequence of the injective property of $\mathscr{C}$ is that in cause-effect chains with undersampling, i.e. where the period of a label's reading task is longer than of its writing task, the result of $\mathscr{C}$ can be the empty set. This is due to the fact that not all instances of the writing task's labels are sampled.

Next, contraints on the latency of cause-effect chains are defined. Note that computing them as exact bounds is shown in Section 4.6.1.

Definition 4.3.6: Minimal/Maximal read-to-write latency

The minimal/maximal read-to-write latency $\mathrm{R2W}^{-}_{i,k}$ / $\mathrm{R2W}^{+}_{i,k}$ along a cause-effect chain $\mathcal{C}_a$ from λ_i to λ_k is defined as the shortest/longest traversal time of all possible cause-effect chain instances.

The maximal read-to-write latency in Definition 4.3.6 corresponds to the notions of *LatencyTimingConstraint* from the AUTOSAR timing extensions in clause 7.2 in [AUT19a]. Latency in AUTOSAR is defined as the time between observable events and categorized into stimuli (as start events) and responses (as end events). The *LatencyTimingConstraint* constrains the minimum and maximum time between a stimulus and possible responses (reaction time), as well es the time from a reaction to possible stimuli (data age). Note that clause 7.3 ("Age Constraint") of [AUT19a] does not apply, since the data path, i.e. the cause-effect chain and its instances (cf. Definition 4.3.3) are explicitly known in SL-LET – clause 7.3 is applied if the sender of data is not explicitly known in an AUTOSAR model. The AUTOSAR Timing Extensions have only an abstract specification which event can serve as stimulus and response events in clause 6 (particularly Table 6.1) of [AUT19a]. Here the following assumptions are made: As responses, the publication times $D^{min}_{i,j}\ \forall j$ of possible data labels from task λ_i are assumed. For stimuli two types of events can be distinguished in the correspondence: (i) the reading event of a task, (ii) the production, i.e. the publication, of data itself. For the latter case two situations have to be distinguished: (a) labels that are written by other SL-LET tasks and (b) sporadically generated data. Fig. 4.4 shows cases (ii)(a) and (ii)(b) and how the time between stimuli and responses can be computed.

In the figure, the write offset Δ_w of two jobs $\lambda_{a,i}, \lambda_{b,j}$, is the time between the earliest publication of data by $\lambda_{a,i}$ and the subsequent read by $\lambda_{b,j}$. For tasks with different rates and offsets from the period the minimum and maximum over all jobs is given by:

$$\Delta^{-}_{w} = \min\{\hat{R}_{b,j} - D^{min}_{a,i}\}\ \forall i,j : \hat{R}_{b,j} \in D_{a,i} \tag{4.8}$$

$$\Delta^{+}_{w} = \max\{\hat{R}_{b,j} - D^{min}_{a,i}\}\ \forall i,j : \hat{R}_{b,j} \in D_{a,i} \tag{4.9}$$

This follows trivially, as the equations simply require that the subsequent job $\lambda_{b,j}$ actually reads from the preceding job $\lambda_{a,i}$. This is the case if the read time is in the data interval of $\lambda_{b,j}$: $\hat{R}_{b,j} \in D_{a,i}$. Δ^{-}_{w} / Δ^{+}_{w} are simply lower and

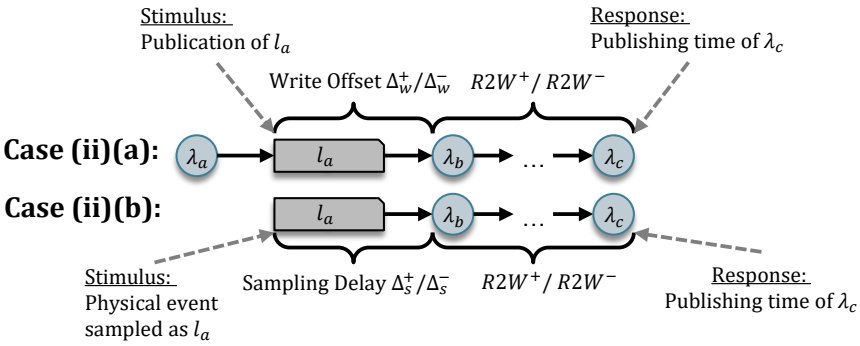


Figure 4.4: Computation of distance between stimulus and response depending on the assumptions on the event type

upper bounds on all possible instances. How these can be computed see Section 4.6.1.

The maximum sampling delay Δ_s^+ is determined by the period P_b of the sampling task λ_b:

$$\Delta_s^+ = P_b \tag{4.10}$$

as this is the maximum time the physical event can actually occur before sampling. The minimum sampling delay is 0, since sampling can happen instantaneous with the physical event that is being sampled. Based on these assumptions the correspondences are given in Table 4.1.

Table 4.1: Correspondence of AUTOSAR Timing Extensions with latency requirements in SL-LET

AUTOSAR Constraint	Case (i) - read event	Case (ii)(a) - label write	Case (ii)(b) - sampling
min. reaction time	$R2W^-$	$R2W^-$	$R2W^-$
max. reaction time	$R2W^+$	$R2W^+ + \Delta_w^+$	$R2W^+ + \Delta_s^+$
min. data age	$R2W^-$	$R2W^+ + \Delta_w^-$	$R2W^+ + \Delta_s^-$

Table 4.1: Correspondence of AUTOSAR Timing Extensions with latency requirements in SL-LET

AUTOSAR Constraint	Case (i) - read event	Case (ii)(a) - label write	Case (ii)(b) - sampling
max. data age	R2W^+	$\text{R2W}^+ + \Delta_w^+$	$\text{R2W}^+ + \Delta_s^+$

4.4 Discussion of SL-LET Model Properties

As pointed out, the SL-LET model introduced in Section 4.2.2 has many ancestors. It is an extension and generalization of the plain LET idea introduced in Giotto [HHK03] as a programming paradigm. Here, SL-LET is used as a design model for timing behavior, particularly the timing of dataflow. The general idea of introducing interconnect tasks to describe communication, where the zero-time communication assumed in plain LET is impossible, stems from [EKG18]. However, the SL-LET model introduced in Section 4.2.2 also has some improvements, some of which are discussed here.

Many other current research, picking up on the LET idea, treat LET / SL-LET only as an implementation paradigm, for the execution of tasks. This manifests e.g. in the industrial case study of a combustion engine control system [HDK+17] where LET is only used to conform data output in a specific way but not intended to be the *design model* of the timing behavior of the dataflow. In doing so many aspects of planing the timing behavior, i.e. "how it should behave", is mixed with assumptions on how the system performs execution. To illustrate this we focus on the connection of LET and period. While most of the early works, e.g. [HHK03] or [KS12], do not explicitly forbid LETs larger than the period, it seems to be an implicit assumption in many other works, to exploit benefits of the model, but in lieu of a clear model specification. The gap of implicit assumptions, i.e. what is expected by the SL-LET model and hence from the implementation, is closed by the definitions in Section 4.2.2.

To illustrate this, Fig. 4.5 shows two cases for a task: Case 1 where $LET > P$ and case 2 where $LET \leq P$. The second case, shows the behavior as it is assumed in many works, e.g. [Bec20, Chapter 6], [MSB18], and [BDM+17]. Since $LET \leq P$, there are no overlapping jobs, which has several implications

for the implementation. First, the semantics of task local variables can be neglected, as there can be no two or more jobs that concurrently try to access this data. Second, the currently running job can always work on the same memory pointer for internal data, and publish its results as the final phase of the job, without having to worry about races, re-entrant code, suitable task output buffering, etc. These situations, however, could arise in case 1 on the left of Fig. 4.5, with $LET > P$. In this case, jobs can overlap in their execution, and data shared among jobs can cause data races, etc.

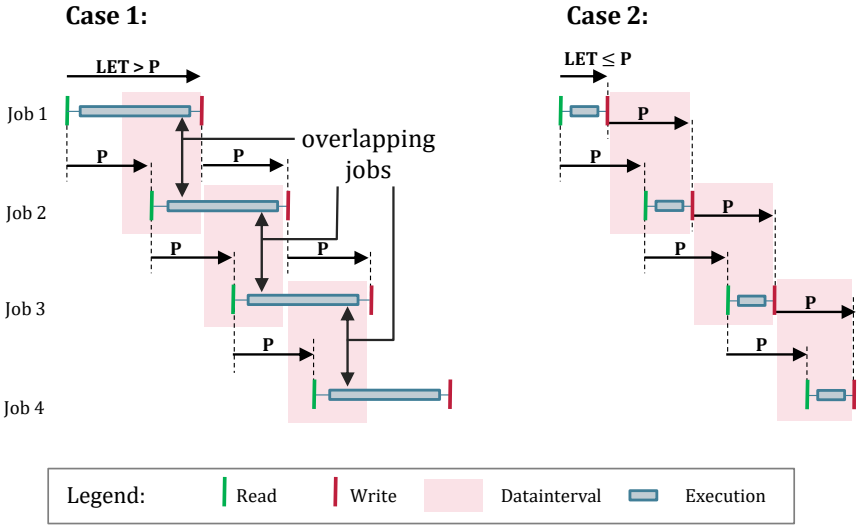


Figure 4.5: Difference in job execution for the two cases: $LET > P$ and $LET \leq P$

From an implementation perspective that focuses on a single task, limiting $LET \leq P$ is convenient, as races are avoided and internal state, that influences the job results, does not have to be documented in the model. From a modelling and specification perspective, this causes trouble, as parameters, such as the LET in this case, abstract too much design information (cf. Section 1.1). A change of the LET in this case is constrained by this hidden knowledge, hence it seems that due to such reasons, previous work has implicitly constrained $LET \leq P$. However, specification of tasks with LETs greater than the period can be reasonable, e.g. for applications where short latencies are not of top priority and higher portability is desired. Since the LET abstracts physical execution, such tasks can easily be ported to platforms where the actual physical execution may take longer than the period,

e.g. in the worst-case. Nevertheless, the dataflow specified in such a (SL-LET) system stays predictable, and the task retains its composability properties (cf. Definition 4.2.1).

In the SL-LET specification given in Section 4.2.2, implicit assumptions are not necessary, even if data shall be preserved across jobs. In this case, the data just ha

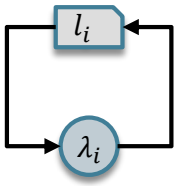


Figure 4.6: Feedback of state information for an SL-LET task λ_i through a label to prevent hidden state of jobs

However, there is a restriction for tasks where state shall be preserved across jobs, but $LET_i > P$. For the task λ_i from Fig. 4.6 consider the following case:

$$LET_i = q \cdot P_i \qquad q > 1, q \in \mathbb{R}^+ \tag{4.11}$$

If the next job $\lambda_{i,j+1}$ of a random job $\lambda_{i,j}$ shall read the label instance $_{i,j}$, it must hold that:

$$D_{i,j}^{min} \overset{!}{\leq} \hat{R}_{i,j+1} \overset{!}{\leq} D_{i,j}^{max} \tag{4.12}$$

Inserting the values from Definition 4.2.4 shows that $\forall q > 1, q \in \mathbb{R}^+$:

$$\hat{R}_{i,j+1} < D_{i,j}^{min} \tag{4.13}$$
$$j \cdot P_i + O_i < (j - 1 + q)P_i + O_i \tag{4.14}$$

Hence, subsequent jobs can not read the state data, since the jobs are *logically* overlapping. By solving Eq. (4.12) for an arbitrary subsequent job $\lambda_{i,j+n}$ instead of $\lambda_{i,j+1}$, the first job to which job data can be feed back, can be computed based on a given q.

The last notable improvement over [EKG18] named here is the introduction of the offset parameter in Definition 4.2.4, which is e.g. also adopted in [GKEQ21]. The offset O_i for a task λ_i has two effects w.r.t. specifying timing behavior. First it implicitly creates a fixed but arbitrary zero-point in the time domain, which allows addressing individual jobs, and allows to reason about the dependencies of jobs different tasks. Second, and most notably from a "user perspective", the offset parameter allows to change data dependencies in cause-effect chain instances by tuning the minimum/maximum read-to-write latency. While, e.g. [GKEQ21] sees the offset as a parameter mainly as an instrument to tune the read-to-write latency, its actual impact is on the compositions of cause-effect chain instances. For illustration, Fig. 4.7 shows two cases for a cause-effect chain $\mathcal{C}_1 = (\lambda_a, \lambda_b)$ where both tasks execute with the same period. Case 1 is shown on the left, where the offsets of both tasks are $O_a = O_b = 0$, and a second case on the right, where the offset of λ_b is set to $O_b = LET_a$. In the second case on the right, the cause-effect chain instance $\mathcal{C}_{1,1} = (\lambda_{a,1}, \lambda_{b,1})$ is different from the one in case 1 on the left, i.e. $\mathcal{C}_{1,1} = (\lambda_{a,1}, \lambda_{b,2})$. However, the instance on the right, has a significantly lower $\mathrm{R2W}^{+}_{a,b}$.

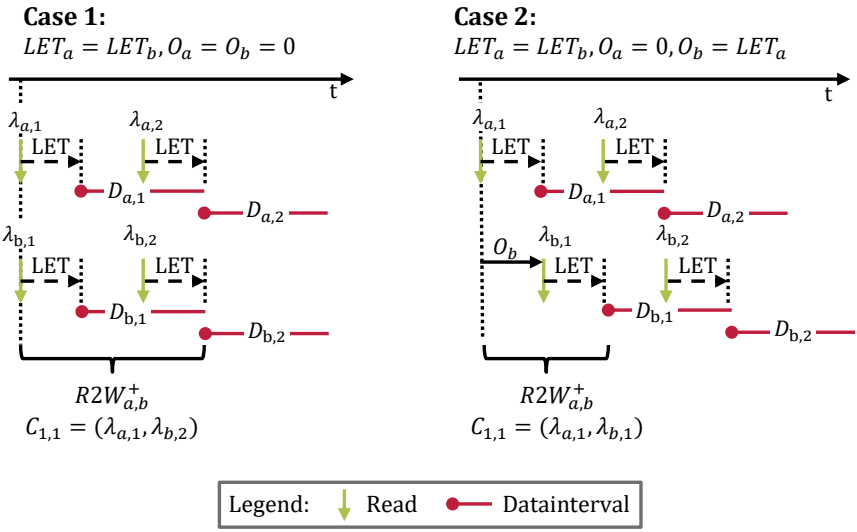


Figure 4.7: Reduction of the read-to-write latency by the introduction of the offset parameter for a dataflow from λ_a to λ_b

4.5 Mapping SL-LET to BET-based Execution

SL-LET bridges a gap that BET-based timing descriptions can not fill in system design. It allows a predictable description of the timing of data-flow along cause-effect chains, more precisely along the instances of cause-effect chains. While in SL-LET cause-effect chain instances can be described at design time by injective relations (cf. Theorem 4.3.4), this is impossible in the BET model. As a consequence, a deterministic timing behavior of the data-flow can not be provided in BET. This is due to the fact, that in BET the read behavior of jobs is heavily influenced by the write behavior of the preceding task in a cause-effect chain, which can not be explicitly specified due to the response-time jitter. Depending on the jitter, multiple scenarios exist from which job the succeeding task in the cause-effect chain reads data. Even if a read-execute-write scheme as in LET and SL-LET is assumed for a BET task τ_i, i.e. it makes a job-local copy of the data it reads, the time when "the read" is actually performed can only be bounded to an interval. This read interval conservatively spans from the activation of the job, as the earliest time the read is performed, to the WCRT minus the BCET $(R_i^+ - C_i^-)$, as the latest possible time the job can perform the read. An injective relation that defines cause-effect chains is hence impossible, as multiple possibilities for data dependencies between jobs exist. In the BET model it is thus a-priori not decidable how cause-effect chain instances look like, or at least shall look like. SL-LET fills this gap.

However, SL-LET can not as straightforwardly be implemented as a BET description of a system. Yet, it is possible to apply SL-LET as a design model in systems, where physical execution follows a BET execution semantic. This is due to the fact, that SL-LET makes no assumptions on the physical execution. In the BET model the assumption is that the physical execution of a single task τ_i requires a processing time between the BCET and the WCET estimates. Scheduling effects such as preemption and blocking cause a response time of every job $\tau_{i,j}$ that is between a BCRT and a WCRT bound. Pragmatically, the response time bounds in BET can be used as an interface to the LET paradigm, which requires that the computation of an (LET) task λ_i is complete no later than the LET_i. Hence, it is possible, to implement the execution phase of λ_i as a BET task τ_i, if the WCRT is less or equal to the LET, i.e. $R_i^+ \leq LET_i$. Obviously, this assumes that a job $\tau_{i,j}$ reads the data intended for the corresponding LET job $\lambda_{i,j}$. The property that data is read and published at the time dictated by the SL-LET model must be supported by the run-time software as the task itself is not able to guarantee it. The literature describes different implementations how reading and publishing data to

readers can be achieved in the context of SL-LET / LET, e.g. to provide buffers for computed results before they are allowed to be published according to the SL-LET design model. [BE18] presents a lightweight approach based on double-buffering, [HHM+16] discusses two approaches in the context of the AUTOSAR Classic platform, whereas one of the approaches is based on the E-Machine concept first reported by [Tem07] as an implementation of the

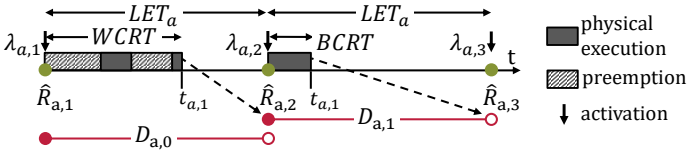


Figure 4.8: BET execution scheme of an LET task where the LET corresponds to the period of the task

Fig. 4.8 shows two jobs of an LET task λ_a and how the SL-LET model corresponds to the BET execution semantic. In the example, the physical execution of the LET task is performed with BET semantics, which implies, that the jobs are activated at $\hat{R}_{a,i}$ and require a processing time between a BCET and WCET. Due to preemption by other tasks through scheduling, each job has a response time between a BCRT and WCRT. These extreme cases are depicted in Fig. 4.8, i.e. the WCRT and BCRT at times $t_{a,1}$ for the first job and $t_{a,2}$ for the second job, respectively. In the BET model, a job's results would be available, i.e. published, as soon as the physical computation has been completed. To comply with the SL-LET paradigm, the label instances, i.e. the data, must not be published before $D^{min}_{a,1} = \hat{R}_{a,2}$ and $D^{min}_{a,2} = \hat{R}_{a,3}$, respectively. For a consuming task, this implies that between $t_{a,1}$ and $\hat{R}_{a,2}$ (which is identical to $D^{min}_{a,1}$ here due to $LET_a = P_a$), the label instance from job $\lambda_{a,0}$ has to be consumed. This circumstance has to be ensured by the run-time software.

The correspondence between the LET of a task λ_i and the response-time bounds of a BET task τ_i can be further generalized as an interface between the BET implementation model and SL-LET as design model. Therefore, we need to consider the correspondences between interconnect tasks as well as normal LET tasks with the BET model in more detail. W.r.t. both models, SL-LET's resources $Res \in \mathcal{R}$ of a platform $\mathcal{P}$ directly correspond to the resources from the BET model, meaning there exists a 1 to 1 mapping between the two. The correspondence rule between SL-LET and BET tasks is

that any LET task λ_i can correspond to a sequence of BET tasks $(\tau_j, \ldots, \tau_{j+k})$ where the sequence is a vertex walk implying event precedence in the BET task graph (cf. Definition 2.1.9). The only limitation is that the BET task chain *must not* cross any timezone boundary. In order to have a valid correspondence, the end-to-end latency, i.e. the WCRT R^+ of the BET task chain must be less or equal to LET_i of the LET task λ_i. The WCRT bound implies activation and termination traces $\omega_{j+n}(n) - \sigma_i(n) \leq R^+ \; \forall \; n \; \in \mathbb{N}_0^+$.

Interconnect tasks, as generalized LET tasks require special treatment in this context, as they are specifically designed to cross timezone boundaries. The correspondence rule for an interconnect tasks Φ_i is therefore identical to that of a plain LET task, with the exception, that the chain may cross a time zone boundary. To account for the synchronization error between time zones, it is assumed that the synchronization error ϵ is part of LET_i. Hence, it is required that $R_i^+ + \epsilon \leq LET_i$.

Fig. 4.9 depicts the correspondence of a SL-LET cause-effect chain $\mathcal{C} = (\lambda_i, \Phi_j, \lambda_k)$ to a BET implementation, where Φ_j is implemented as an Ethernet communication. In the example the communication resources Com_1 and

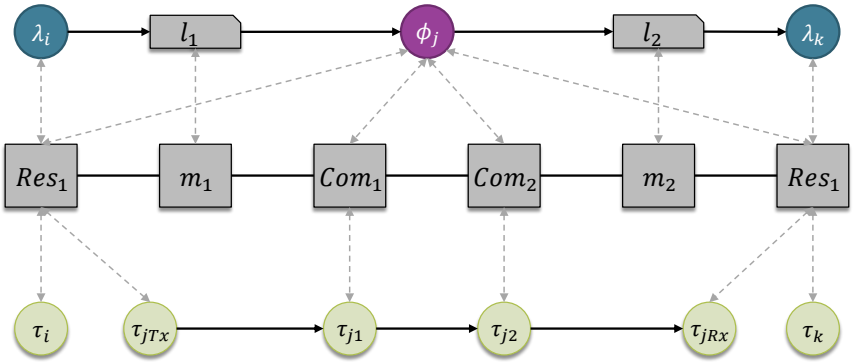


Figure 4.9: Example for the correspondence of an LET cause-effect chain to a BET chain

Model correspondence between SL-LET and BET model is formally expressed by a number of relations, which assigns properties from one model to a property of the other and vice versa. First of all, model correspondence

requires that events in the SL-LET model are assigned to observable events in the BET model. This is defined on a per task basis:

Definition 4.5.1: Task Event Correspondence

An LET task or interconnect task $\lambda_i \in \Lambda$ corresponds to a chain of BET tasks $w = (\tau_a, \ldots, \tau_b)$, where w is a vertex walk in $\mathcal{TG}$, through relations:

$$\text{map}_i^{rd} = \{(\sigma_i^{LET}(n), \sigma_a(n))) \mid n \in \mathbb{N}_0^+\}$$
$$\text{map}_i^{wr} = \{(\omega_i^{LET}(n), \omega_b(n)) \mid n \in \mathbb{N}_0^+\}$$

where map_i^{rd} defines a *read event correspondence* between λ_i and τ_a, and map_i^{wr} defines a *write event correspondence* between λ_i and τ_b.

Hence, a task event correspondence relates the read events from the SL-LET model to activation events in the BET model, as well as publication events in the SL-LET model to termination events in the BET model. Done for all tasks in the SL-LET model, it results in correspondence for the entire model:

Collorary 4.5.2: Model Correspondence

The correspondence for the entire model is achieved by mapping all tasks $\lambda_i \in \Lambda$ from the SL-LET model:

$$\text{MAP}^{rd} = \bigcup_{\forall \lambda_i \in \Lambda} \text{map}_i^{rd}$$
$$\text{MAP}^{wr} = \bigcup_{\forall \lambda_i \in \Lambda} \text{map}_i^{wr}$$

In order to form a *valid* correspondence, three requirements must be fulfilled. First, it is assumed that the run-time environment buffers data, such that it is available to consumers for the entire data interval of its producer. Based on this assumption the chronological order of events must ensure that data is read when the intended data are available and that output data is ready for publication by the run-time before the data interval starts. It is assumed that BET tasks consume input data with their activation and produce output data

with their termination. This is referred to as *read validity* and *write validity* and is illustrated for a job $\tau_{a,1}$ that corresponds to $\lambda_{a,1}$ in Fig. 4.10.

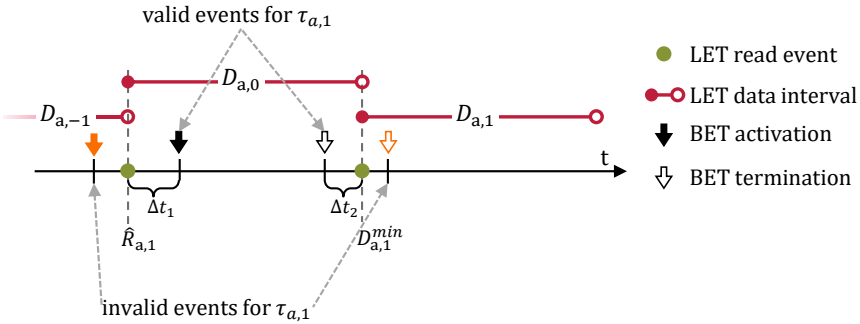


Figure 4.10: Valid chronological order of events for job $\tau a, 1$

For job $\tau_{a,1}$ two possible activation events and two possible termination events are shown. For the correspondence to be valid the activation event $\sigma_a(1)$ must occur after the read event, i.e. $\sigma_a(1) \geq \hat{R}_{a,1} = \sigma_a^{LET}(1)$. This only holds for the event Δt_1 after $\hat{R}_{a,1}$, indicated in black in Fig. 4.10. The orange activation event shown in orange, however, would reside in the previous data interval $D_{a,-1}$ and thus is *invalid* as the job could read the wrong data upon its activation.

Formally read and write validity for all jobs are defined by:

Theorem 4.5.3: Read Validity

The chronological order of read events is valid for a task λ_i with read event correspondence map_i^{rd} iff:

$$t_a \leq t_b \qquad \forall\, e = (t_a, t_b) \in \text{map}_i^{rd}$$

Theorem 4.5.4: Write Validity

The chronological order of write events is valid for a task λ_i with write event correspondence map_i^{wr} iff:

$$t_a \geq t_b \qquad \forall\, e = (t_a, t_b) \in \text{map}_i^{wr}$$

Assuming that the LET run-time software buffers data accordingly as e.g. described in [BME16] to avoid that job $j+1$ already overwrites data of job j which are still published.

Hence, a SL-LET model and a BET model corresponds, iff read validity and write validity can be proven for all task event correspondences in MAP^{rd} and MAP^{wr}.

For illustration, let us consider the example from Fig. 4.9. The read correspondences are given by:

$$\text{map}_i^{rd} = \{\sigma_i^{LET}(n), \sigma_i(n) \mid n \in \mathbb{N}_0^+\} \tag{4.15}$$
$$\text{map}_{jTx}^{rd} = \{\sigma_j^{LET}(n), \sigma_{jTx}(n) \mid n \in \mathbb{N}_0^+\} \tag{4.16}$$
$$\text{map}_k^{rd} = \{\sigma_k^{LET}(n), \sigma_k(n) \mid n \in \mathbb{N}_0^+\} \tag{4.17}$$

and the write correspondences by:

$$\text{map}_i^{wr} = \{\omega_i^{LET}(n), \omega_i(n) \mid n \in \mathbb{N}_0^+\} \tag{4.18}$$
$$\text{map}_j^{wr} = \{\omega_j^{LET}(n), \omega_{jRx}(n) \mid n \in \mathbb{N}_0^+\} \tag{4.19}$$
$$\text{map}_k^{wr} = \{\omega_k^{LET}(n), \omega_k(n) \mid n \in \mathbb{N}_0^+\} \tag{4.20}$$

In the example, it is assumed that all resources are scheduled priority driven. The priority (lower value indicates higher priority) and the event models are provided in Table 4.2.

Table 4.2: Event models for the example system depicted in Fig. 4.9

BET Task	**Priority**	**Event Model**
τ_i	1	$\delta^-(n) = (n-1) \cdot P_i$
τ_{jTx}	2	$\delta^-(n) = (n-1) \cdot P_j$
τ_k	1	$\delta^-(n) = (n-1) \cdot P_k$
τ_{j1}, τ_{j2}	1	propagated (task is event dependent on a predecessor)
τ_{jRx}	2	propagated (task is event dependent on a predecessor)

Let us focus on the write validity of Φ_j. Since the event model is an upper/lower bound on any activation trace:

$$(\delta^-_{jTx}(n), \delta^+_{jTx}(n)) \models \sigma_{jTx} \tag{4.21}$$

we use its properties for the proof of write validity:

Proof 4.5.5:
In order to fulfill Theorem 4.5.4 it needs to be proven that for any event in all possible traces:

$$\omega_j^{LET}(n) \overset{!}{\geq} \omega_{jRx}(n) \qquad \forall n \in \mathbb{N}_0^+ \tag{4.22}$$

holds. The time from activation of τ_{jTx} to the termination event of the chain of τ_{jRx} can be expressed by the response time $R(n)$ for any $n \in \mathbb{N}_0^+$ as:

$$R(n) = \omega_{jRx}(n) - \sigma_{jTx}(n) \tag{4.23}$$
$$\omega_{jRx}(n) = \sigma_{jTx}(n) + R(n) \tag{4.24}$$

Since the activation of τ_{jTx} is strictly periodic it follows:

$$\omega_{jRx}(n) = n \cdot P_j + R(n) \tag{4.25}$$

Furthermore, the response time can be upper bounded $\forall n \in \mathbb{N}_0^+$ by the WCRT R^+:

$$\omega_{jRx}(n) \leq n \cdot P_j + R^+ \tag{4.26}$$

Under the assumption that $LET_j \geq R^+$ it follows:

$$\omega_{jRx}(n) \leq n \cdot P_j + LET_j \tag{4.27}$$
$$\omega_{jRx}(n) \leq n \cdot P_j + LET_j = \omega_j^{LET}(n) \tag{4.28}$$
$$\omega_{jRx}(n) \leq \omega_j^{LET}(n) \tag{4.29}$$

□

Similarly, the proofs for write validity of λ_i and λ_k can be obtained, but are omitted for brevity.

The read validity of Φ_j can be trivially proven due to the strictly periodic event models:

Proof 4.5.6:
In order to fulfill Theorem 4.5.3 it needs to be proven that for any event in all possible traces:

$$\sigma_j^{LET}(n) \overset{!}{\leq} \sigma_{jTx}(n) \qquad \forall n \in \mathbb{N}_0^+ \tag{4.30}$$

This follows trivially by inserting the definition of $\sigma_j^{LET}(n)$ and the properties of $\delta_{jTx}^-/\delta_{jTx}^+$:

$$n \cdot P_j \leq n \cdot P_j \tag{4.31}$$

□

Similarly, the proofs for read validity of λ_i and λ_k can be obtained, but are omitted for brevity.

In both proofs 4.5.5 and 4.5.6 we have seen that the assumption $R^+ \overset{!}{\leq} LET$ holds is essential. However, the length of an LET interval is a design parameter that can be controlled by a system designer. Obviously, LETs can not be chosen arbitrarily as the system has to fulfill other constraints as well, e.g. data-age constraints along a given (functional) data path. Through the cross-layer model such constraints can be transformed into constraints in the SL-LET model as well. Hence, in the following subsection it is investigated how SL-LET designs can be verified such that the design meets such constraints. Furthermore, also the **dependencies** of verified designs are investigated. Since the verification results are only valid iff, the dependencies that the design and the incurring correspondence entail are acceptable in the sense of the safety concept. Again the focus is on timing dependencies and what safety measures must be imposed on the design to accept the design as *safe* w.r.t. top-level functional safety requirements.

4.6 Verification and Validation of SL-LET Designs

4.6.1 Verification of End-to-End Latency Requirements in SL-LET Designs

In contrast to the BET timing model, SL-LET allows computing end-to-end latencies for a cause-effect chain in a composable manner (cf. Definition 4.2.1). The analysis that is introduced here, is a generalized idea for timezone-local cause-effect chains as already described by e.g. [BDM+18],[BDM+17], and [MSB18] and also extended to LET tasks with offsets. The offset is used, such that read events of a succeeding task in a cause-effect $\mathcal{C}_i$ coincide with the beginning of a data interval. In these cases, the read-to-write latency can be computed as the sum of the LETs of both tasks. For generalization to the system-level idea of SL-LET, where interconnect tasks with an arbitrary LET, form the bridge between different time zones, it needs to be considered that an alignment of read event with the beginning of a data interval is not always the case, i.e. that a reading task performs the read event at an arbitrary point in the data interval. Note that, this extension works for all offset combinations between interconnect tasks and normal LET tasks : $\Phi_i \rightarrow \lambda_j, \lambda_j \rightarrow \Phi_i, \lambda_i \rightarrow \lambda_j$ and $\Phi_i \rightarrow \Phi_j$.

Theorem 4.6.1: Computing Minimum/Maximum read-to-write latency

The minimum/maximum read-to-write latency for a cause-effect chain $\mathcal{C}_i = \{\lambda_a, ..., \lambda_b\}$ can be found among the cause-effect chain instances where the jobs of λ_a reside in the same hyperperiod (i.e. $lcm(P_a, ..., P_b)$) of all tasks in $\mathcal{C}$.

Proof 4.6.2:
The hyperperiod as the least-common multiple of the periods of all tasks in the cause-effect chain denotes the time intervall after which the cause-effect chain instances start to repeat themselfs, merley shifted in their job indices. Since cause-effect chains are formed injectively it is further fact that for two jobs $\lambda_{a,i}$ and $\lambda_{a,j}$ with $i \neq j$, the resulting set of reachable jobs is disjunct, i.e. $\mathscr{C}(\lambda_{a,i}) \cap \lambda_{a,j} = \emptyset$. Hence, the maximum and minimum latency must be found in one of the sets $\mathscr{C}(\lambda_{a,i})\ \forall i :\ \hat{R}_{a,i}$ is in the same hyperperiod.

□

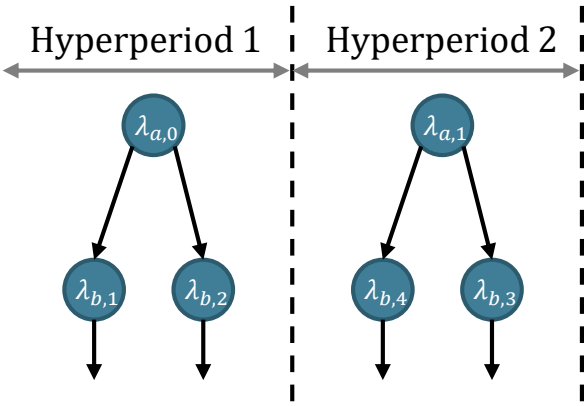


Figure 4.11: Reachability Graph for the LET effect chain depicted in Fig. 4.3 from λ_a to λ_b

More illustratively, the possible data propagation paths, i.e. cause-effect chain instances, can be collected in the data propagation graph $\mathcal{G}(\mathcal{C}_x)$ for a cause-effect chain $\mathcal{C}_x$. A node in $\mathcal{G}(\mathcal{C}_x)$ represents a job of one of the tasks $\lambda_i \in \mathcal{C}_x$. A directed edge $(\lambda_{i,j}, \lambda_{k,l})$ then indicates data flow from $\lambda_{i,j}$ to $\lambda_{k,l}$, i.e. $\hat{R}_{k,l} \in D_{i,j}$.

4.6.1.1 Constructing Job Reachability Graphs

The data propagation graph of all neccessary cause-effect chain instances according to Theorem 4.6.1 can be generated by a recursive approach. Algorithm 5 is executed for all jobs $\lambda_{i,j}$ of the first task in a cause-effect chain. However, only the jobs lying in the first hyperperiod of the system must be considered as initial jobs, since the activation sequence and thus the system's timing behavior repeats in the next hyperperiod. This is due to the fact that in the SL-LET model interconnect tasks as well as normal LET tasks are strictly periodic. Note that a job that is reading from $\lambda_{i,j}$ might lie in the next hyperperiod. Nevertheless, the data propagation path is considered. As a consequence, all possible data propagation paths are then contained within $\mathcal{G}(\mathcal{C}_x)$.

In Algorithm 5 lines 1 to 3 determine whether the processed job is an instance of the last task in the cause-effect chain. If this is the case no further successors exist, and the recursion can terminate. Yet, if it is not, the jobs of

Algorithm 5: process_job($\mathcal{C}, \lambda_{i,j}$)

```
1 λ_k ← get_successor(C, λ_i);
2 if λ_k = ∅ then
3 |   return
4 for λ_{k,l} in find_readers(λ_k, λ_{i,j}) do
5 |   add_edge(λ_{i,j}, λ_{k,l});
6 |   process_job(C, λ_{k,l}) ;
7 end
```

the succeding task λ_k that potentially read from $\lambda_{i,j}$ must be found. This is achieved by *find_readers()* in line 4. The function returns all jobs $\lambda_{k,l}$ such that $\hat{R}_{k,l} \in [D_{i,j}^{min}; D_{i,j}^{max}($. All $l \in \mathbb{N}_0^+$ with the first job $l = 1$ starting/reading at time $t = 0$ can be found by the following equation:

$$D_{i,j}^{min} \leq \hat{R}_{k,l} < D_{i,j}^{max} \tag{4.32}$$

$$(j-1) \cdot P_i + O_i + LET_i \leq (l-1) \cdot P_k + O_k < j \cdot P_i + O_i + LET_i \tag{4.33}$$

For each $\lambda_{k,l}$ an edge from $\lambda_{i,j}$ is inserted in the data propagation graph – whereas $\lambda_{k,l}$ or $\lambda_{i,j}$ can be an interconnect task as well. The weight w of this edge is set to the read-to-read distance $\text{R2R} = \hat{R}_{k,l} - \hat{R}_{i,j}$. This is necessary to compute the end-to-end latency in the next step. Line 6 starts the recursion. Since cause-effect chains are already acyclic graphs, also the resulting data propagation graph of a cause-effect chain is cycle free.

4.6.1.2 Computation of Latencies

Since for each cause-effect chain $\mathcal{C}_x$ Algorithm 5 is executed for all initial jobs in a hyperperiod, the data propagation graph $\mathcal{G}(\mathcal{C}_x)$ contains all possible data propagation paths starting in this hyperperiod, i.e. all cause-effect chain instances. Due to the periodic nature of the system, $\mathcal{G}(\mathcal{C}_x)$ also contains the worst case. For each path in $\mathcal{G}(\mathcal{C}_x)$ the read-to-write latency can be computed by summing up the edge weights w (read-to-read latencies R2R) plus the execution of the last task λ_n in $\mathcal{C}_x$, i.e. LET_n. Note that the read-to-read

latency of adjacent tasks λ_i and λ_{i+1} in $\mathcal{C}_x$ also contains the execution of λ_i, i.e. LET_i.

Due to the acyclic nature of the job reachability graph, the shortest and longest read-to-read path based on the edge weights can conveniently be found using standard graph algorithms [SW11]. The minimum and maximum read-to-write latency R2W_x^- and R2W_x^+ are thus the shortest and the longest path plus the additional LET of the last task in $\mathcal{C}_x$ respectively.

4.6.2 Timing Dependency Analysis and Monitoring Synthesis for Validation of SL-LET in BET Execution Semantics

So far it has been shown how SL-LET allows designing cause-effect chains composable and how to verify design requirements like end-to-end latency in the design model. Investigating only the SL-LET design model for (timing) dependencies as done in Chapter 2 shows that no timing dependencies between chains in any $\mathcal{LG}$ exist. This is exactly the composability property described by Definition 4.2.1 that is expected from SL-LET. However, we have seen in Chapter 2 that abstraction of a system design can hide dependencies. The abstraction that SL-LET performs w.r.t. the design, can be overcome by taking the implementation model, i.e. the BET model into account again. While, separating the two models at first might seem odd, we shall see that it makes sense to separate concerns this way. The design model fulfills the purpose of describing a predictable timing behavior and verifying timing requirements imposed by the functional model on it. The implementation model on the other hand, is better suited to describe the timing behavior at run time. The model correspondence introduced in Section 4.5 allows to connect the two, i.e. lift the abstraction in the (SL-LET) model for dependency analysis purposes. This allows to investigate whether the *"guarantees"* made by the verification of the design model hold in the implementation. Following the design idea of cross-layer dependency analysis, it is possible to validate whether the assumptions the design model imposes on the implementation are fulfilled in the implementation model. Note, that the focus here is solely on the timing behavior, i.e. whether there is (bounded) dependence on timing parameters of the BET implementation. The implementation of SL-LET must still guarantee correctness of functional properties of SL-LET, in order to guarantee the timing of the data-flow along cause-effect chains. The existing approaches for communication can be categorized into solutions which require explicit buffer management,

as e.g. in [BE18] or one of the variants in [HHM+16], and implicit buffer management. In the implicit variant, tasks must fulfill the necessary timing behavior and provide the results to memory according to cause-effect chain instance definitions. This avoids explicit synchronization and synchronizes only implicitly via the expected run-time timing behavior, i.e. writing and reading to appropriate memory locations depending on job sequence numbers. Such a solution is e.g. possible via AUTOSAR CP OSScheduleTables, as hinted by [HHM+16]. In any case, a validation for the timing behavior of the corresponding BET model through timing dependency analysis is valuable as correct timing behavior is required for all communication methods – implicit and explicit.

The proofs of both Theorem 4.5.3 and Theorem 4.5.4 require the assumption that the WCRT of a BET chain is less or equal to the LET for a valid model correspondence (cf. Section 4.5). Hence, WCRTs in the BET model become the first object of investigation for the validation of an (SL-LET) design. Dependency analysis here is used to identify and trace on which BET timing parameters the guarantees in the SL-LET model depend. Seen from the perspective of requirements tracing, the assumptions in a correspondence proof become requirements in the implementation model, which in turn also need verification.

Section 2.5 introduced how TDGs can be used to reveal the dependencies of a timing requirement in BET semantics. Hence, validation of the SL-LET to BET model correspondence in the form of a dependency analysis, consists of four steps:

1. Importing the assumptions the LET-BET model correspondence makes as requirements in the BET model
2. Verification of these requirements through suitable timing analysis
3. Timing dependency analysis of the results of the timing analysis
4. Confidence analysis of the timing dependencies

For illustration, consider the example depicted in Fig. 4.12, which extends the example from Fig. 4.9 by a second cause-effect chain $\mathcal{C}_2$. In this example, a predictable timing behavior of the data-flow along $\mathcal{C}_1 = (\lambda_i, \Phi_j, \lambda_k)$ is the goal. Each LET chain implements a different functionality for which freedom from interference is required. The first step sets the LETs of all corresponding BET task chains as the deadline for the scheduling analysis conducted on the BET model. Verifying that these deadlines are met is the second step, which can be carried out e.g. by CPA. Let us consider the

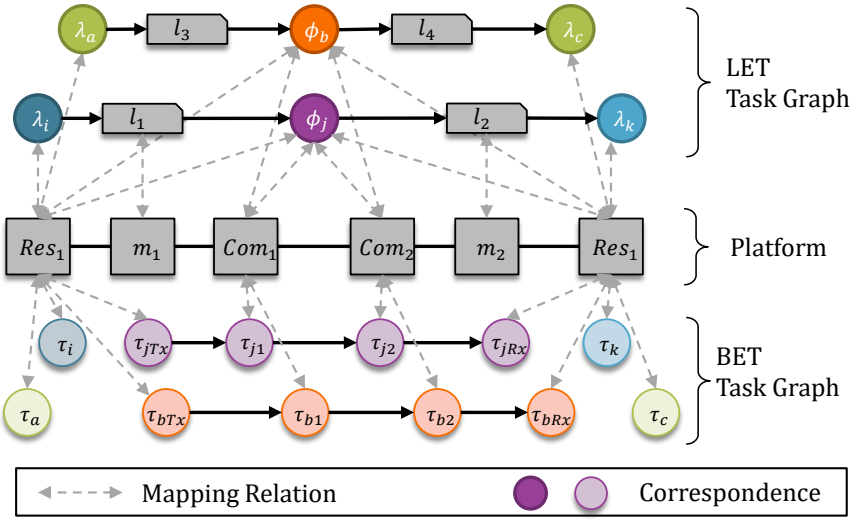


Figure 4.12: Example System with two LET cause-effect chains with corresponding BET tasks

LET cause-effect chain $\mathcal{C}_1 = (\lambda_i \rightarrow \Phi_j \rightarrow \lambda_k)$. Computation resources Res_x are scheduled with SPP and communication resources Com_y are scheduled with SPNP. The scheduling priorities for the BET tasks are chosen such that all possible BET tasks interfere with the corresponding BET tasks (i.e. $\xi_1 = (\tau_i, \tau_{jTx}, \tau_{j1}, \tau_{j2}, \tau_{jRx}, \tau_k)$ of $\mathcal{C}_1$. Further, assume that the timing analysis computes WCRTs that are all less or equal to the required deadlines, i.e. the LETs. The resulting TDG (which is not shown here due to its complexity and space reasons) reveals the timing dependencies between the BET tasks that implement $\mathcal{C}_1$ and $\mathcal{C}_2$. As all the scheduling parameters are chosen such that all BET tasks corresponding to $\mathcal{C}_2$ interfere with the BET tasks of $\mathcal{C}_1$, consequently all WCRTs of $\mathcal{C}_1$ depend on values of execution times and event models of the BET tasks of $\mathcal{C}_2$. In Section 2.5 it has been shown that dependence can be acceptable as bounded timing dependence (cf. Definition 2.5.5), if sufficient confidence exists into the timing parameters on which a timing analysis result depends. Furthermore, in Section 3.2 techniques to increase the confidence of timing properties of the BET model through run-time monitoring and enforcement have been presented. In conclusion, timing dependency related issues arising from the BET implementation can be solved by the named methods and validation can be passed successful.

The original idea behind designing with SL-LET is that along cause-effect chains the timing of the data-flow can be designed independent of other chains sharing the same system platform. Accepting *bounded dependence* in a BET implementation of an (SL-LET) design, the idea becomes feasible. For a validation of bounded dependence, however, it is required that no run-time faults occur which would cause that any timing parameter will be exceeded beyond its specified bounds. Only under this assumption the system specified by the CLM will exhibit run-time behavior that is compliant with the behavior expected from the model. There is clear differentiation between run-time faults, where the root cause occurs stochastically at run-time, and a design fault, where the root cause is at design time, i.e. before run-time. The design fault at hand is that insufficient model knowledge results in a non-conservative WCET estimate and in consequence that a bound which is specified in a timing parameter is violated at run-time. Hence, it is not a run-time fault, although the effects (i.e. the error) manifest at run-time.

To prevent either the design fault as a whole or to contain it at run-time under the assumption that no further run-time faults appear, two approaches have been introduced. Either to allocate safety requirements for safety process measures to prevent the design fault by requiring more (and conservative) knowledge about the system (cf. Section 3.1) , or to perform monitor synthesis to enforce the model correspondence (cf. Section 3.2).

However, we have also seen in Section 3.3 that a successful and efficient use of monitors heavily depends on well configured monitors. A well done configuration requires knowing when deadlines or other extra-functional timing requirements would be violated to reject such a configuration in the course of the sensitivity analysis. In the related work on sensitivity analysis presented in Section 3.3 we have seen that sensitivity analysis heavily relies on *deadlines* for all tasks. While solely with the BET model as design and implementation model this timing requirement was hard to provide by function developers, this circumstance changes with the introduction of SL-LET as design model. Even more troublesome is the circumstance that the specification of a deadline might even be a “shared” requirement between function engineer and integration engineer, as the deadline can be used to “tune” the timing of the data-flow. With SL-LET specifying a predictable data-flow is possible explicitly. Through the correspondence rule with the BET model, deadlines in the form of the LET label-write time become available for the sensitivity analysis in the BET model. An advantage that response-time requirements that originate from the correspondence have is, that they are specified for independent tasks. Sensitivity analysis

is no longer dependent on timing bounds as "deadlines" that are based on intervals for entire chains as e.g. [SMT+18], [BDM+16] and [BDM+17] provide them if only maximum data-age requirements are specified for BET cause-effect chains that communicate sampling based, i.e. without event dependence in the BET model. In consequence, the LETs are a perfect "deadline" specification, as they are also easy to use in the TDG based monitor synthesis, while maintaining predictable data-flow if a suitable run-time mechanism is in place. BET on the other hand can only maintain deadlines without well predictable properties, i.e. cause-effect chain instances, of the data-flow.

4.7 Conclusion

One can now ask what the gain of introducing a separate design model and additional verification and validation is. The first gain is that certain concerns have been separated in individual models and techniques.

Foremost, SL-LET is a perfect way of specifying the timing of data-flow, even in complex cause-effect chains traversing multiple resources of a system. This circumstance is impossible with BET-based timing description of cause-effect chains. Although, it has to be noted that in order to provide the predictable timing of data-flow, a faithful SL-LET implementation can require additional run-time mechanisms to prevent deviations due to late reads or early writes, as mentioned above.

Second, SL-LET is a requirements engineering tool for the dependency identification and mitigation in the BET implementation model. It provides clear requirement towards the BET model, while at the same time allowing the function designer to work with composable timing. Particularly, it provides discrete valued results of an end-to-end timing analysis to the function developer, compared to intervals with undecidable behavior in them.

The third benefit resulting from the separation of concerns is that it allows a better understanding of errors and possible failures of a design – both at run time and at design time. Understanding failures and error effects ultimately can improve safety of the system, as it allows introducing or tune counter measures (including safety process measures). Now the concrete benefit is with rather complex types of errors. Avizienis et al. [ALRL04, Fig.8] differentiate in their taxonomy that data can either be wrong, have incorrect timing, or both. As an example, consider a cause-effect chain with sampling

at different rates. If offsets are introduced to prevent peak load, longer response times of jobs can easily cause errors of the last class, where due to incorrect timing wrong data and thus incorrect service is produced. The determinism of timing introduced with SL-LET prevents this error class in the design model, i.e. at design time, while the TDG based validation limits the probability that run-time behavior of this error class can occur. Note that a faithful BET implementation must not only avoid late writes through the validation of correspondence, i.e. $LET \leq R^{+}$, but also postpone early writes to the publication time of a label according to the SL-LET model. The latter is a property that is achievable with implementations as e.g. provided by [BE18], but not the former. Nevertheless, the question remains, whether verification and validation have become easier and how the safety process can gain from the introduction of (SL-LET) in the cross-layer model for dependency analysis. This is the focus of the next chapter Chapter 5. Particularly, whether or how the assumption that no stochastic run-time faults occur, can be relaxed and still ensure timing safety.

Chapter 5

Timing Diversity as a Safety Measure

So far the focus of this thesis has been to investigate how system designers can apply more stringent design measures in a worst-case design, such that dependencies do not result in destructive interference and ultimately lead to catastrophic consequences. With the focus on timing behavior of a system, all the measures have the intention of avoiding timing interference which might lead to requirements violation. Whereas the interference on a particular application can be self-interference, e.g. by exceeding a specified WCET or arrival pattern specification, or external interference from another application. In this quest for better verification and validation methods it has become obvious, that ever-increasing engineering efforts in conventional specification and analysis methods are insufficient.

The last chapter has introduced a novel design model that refrains from the conventional specification of embedded software by actual execution time demand and occurrence of that demand, in favor of a design specification that focuses on a response-time based specification. This shift was motivated not only by the fact that traditional execution time specification for complex embedded software has become intractable but also because it has significant benefits for specifying requirements that improve functional behavior. While for instance the conventional analysis methods for data-age are only able to verify that a certain lower and upper bound are maintained at runtime, nothing can be said about the jitter and the typically observed data-age [SMT+18][BDM+17][BDM+18]. In comparison to that, SL-LET shifts

the focus from the actual timing of the execution to the timing of data. The system's specification, similarly to the executed functions, becomes more data-centric. The previous section has furthermore shown how the design model can be checked for requirements satisfaction (*verification*) and how correspondence with an implementation model can be checked for correct assumptions by dependency analysis (*validation*). These analyses are conducted under the assumption that the system is fault free, i.e. that all parameters are correctly and conservatively defined. In this section, we lift this restriction to a certain extent. However, the goal of ensuring timing safety of the system persists. The restriction, that is lifted is that the specifications in the implementation model do not need to be ultimately conservative. What is assumed instead is that their *average* worst-case is sufficient to fulfill the SL-LET paradigm, i.e. correspondence with the SL-LET model. This creates a situation where timing errors can occur, e.g. in a worst-case situation that goes beyond the specified average worst-case. Resulting timing errors are masked by *timing diversity* such that the original cause-effect chain can execute as if no timing error had occurred. This allows more flexible designs.

The flexibility gained by the assumption is extended to the concept of *timing diversity* which allows to maintain the timing safety requirements on the data. The next section first introduces the concept, while the subsequent section discusses it from different angles: implementation, synthesis from the initial design-model, and reliability under different fault assumptions. An experiments and case study section shows how the concept increases overall reliability and how the concept can be applied to a state-of-the art environment perception function for automated vehicles. The environment perception function is specifically chosen, as it is a hard or even impossible challenge for the mechanisms used in Chapter 2 and Chapter 3. Finally, Section 5.4.2 presents how the concept can be exploited in the context of ISO 26262 to reduce cost intensive process measures by systematically decomposing requirements according to the concept.

5.1 The Idea of Timing Diversity

In finance, diversifying a portfolio is a measure to reduce the risk that the entire investment gets lost by stretching the investments over multiple stocks. The principle of stretching the risk of failure is also applied in *timing diversity*. The idea of timing diversity assumes that two jobs of a specific task executing the same workload will not exhibit identical execution time

on complex architectures. Hence, the risk that both jobs are affected by a timing error is lower than a timely execution of at least one the jobs. By placing these jobs quasi parallel in time into independent timing channels and selecting a channel which is "on time" w.r.t. the specification, the risk of a timing failure can be mitigated. More precisely, at the end of the LET the channel selection logic in Fig. 5.1 will select the channel which has finished successfully and publish its data to subsequent readers.

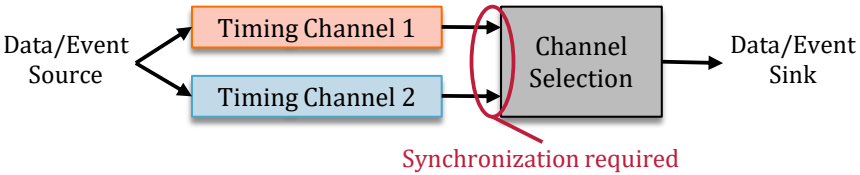


Figure 5.1: Timing Diversity scheme based on homogeneous redundancy

The scheme presented in Fig. 5.1 requires that both channels need to be at least loosely synchronized, i.e. it must be possible to match data or events going into the two channels to output events from the channels. Particularly in entirely BET specified systems with sampling-based communication synchronization poses a problem due to the large jitter between BCRT and WCRT. By specifying and implementing the system based on a SL-LET specification, this problem is solved, as timing of data is explicitly specified. An additional synchronization mechanism but a label instance id is not necessary.

One pillar of the timing diversity scheme is that on complex architectures, the execution time strongly depends on the state of the hardware and software execution platform (caches, branch prediction, memory mapping, kernel state, etc.). This state is not only influenced by the application, i.e. the task, itself but also by every other task executing on the resource and in consequence the state is altered. Due to the vast number of alterations on a comparatively short timescale compared to the overall execution of a job, these are in a way unpredictable or at least undecidable in the available models [CKM+19][DSA+13]. As an example, take a memory access through a cache on an architecture that features an MMU. While the access which might change the state of multiple caches in the hierarchy is in the order of *ns* or a few clock cycles, overall execution of jobs is rather in the order of *ms* [ARM13]. This idea follows the argumentation of [ALRL04] and [GM02], which state that the behavior of a system or component is the result of the underlying structure. The underlying structure in this case is the possible

state space, whereas the resulting observable behavior is the execution time behavior of a job. Timing diversity exploits this fact instead of combating it with even more rigorous engineering methods. One assumed consequence of this vast state space and possible transitions in it is that the probability that two *parallel* instances exhibit the same execution time abnormality from the average case due to platform effects is marginal, even if they operate on the same input data. Referring to a timing error in timing channel 1 as E_{Ch1} and in timing channel 2 as E_{Ch2}, this implies for the probabilities P of the events:

$$P[E_{Ch1} \cap E_{Ch2}] = P[E_{Ch1}] \cdot P[E_{Ch2}] \tag{5.1}$$

which implies that timing errors E_{Ch1} and E_{Ch2} are assumed to be statistically independent.

The possibility to mitigate an error in one channel allows to relax the conservatism in the specification of the individual channels compared to a non-redundant setup under certain conditions. The performance of the timing diversity approach in terms of timing reliability will be discussed in detail later. But in general, it allows to relax the requirements in the specification of the channels, e.g. by reducing the gap between conservative WCET assumption and a value observable in measurement based testing. The problem of non-conservative specification parameters in the design process was already discussed in Chapter 2 and Chapter 3. The proposed solution there is to introduce the notion of confidence for system parameters, like for instance the WCET parameter. The method has the goal of limiting the timing interference to tasks with a sufficiently high confidence level, to compute and bound the interference.

5.2 Related Work

For pure communication instead of computation a similar idea is the basis of the redundancy feature in Avionics Full-Duplex Ethernet (AFDX) as the implementation of ARINC664 Part 7 [Aer09]. In AFDX redundancy is managed per virtual link implemented on two independent switched networks, the A and B Networks. A packet transmitted over a redundant virtual link by an End System is sent on both networks. Therefore, under normal operation, each End System will receive two copies of a packet, if the packet is sent on a duplicated link. The reception logic of the receiving End System is

shown in Fig. 5.2. It follows a "first valid packet wins", which avoids complex synchronization mechanisms. Therefore, it uses one byte as a sequence number. Compared to standard Ethernet, this sequence number resides in the Ethernet payload before the Frame Check Sequence for each Virtual Link to identify redundant copies of a packet. In essence, the idea of timing diversity for computation is very similar to the redundancy idea of AFDX for communication. Similar to timing diversity the end-to-end response time of the same original packet varies in network A and B. The difference to the SL-LET approach of timing diversity is that the Redundancy Management unit of AFDX forwards a correctly received packet immediately and not a predetermined point in time, as only the integrity must be checked. One of the main reasons, why ARINC 664 Part 7 convinced certification bodies of its robustness and safety is that it is a low complexity solution for the single fault assumption where one packet might be lost. As such it was certified for service in several commercial aircraft from Airbus as well as Boeing.

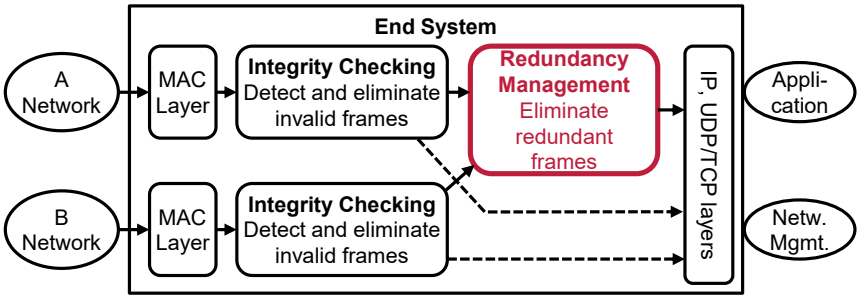


Figure 5.2: AFDX Redundancy Management, Source:Figure 3-16, ARINC 664P7 [Aer09]

W.r.t. automotive architectures a number of fail-safe platforms have been proposed. However, the ones reported in the literature do focus on other aspects of fail-operational capability. For instance, based on the platform for the RACE project [SCB+13], [BSBA14] computes fail-operational deployments of software based on hardware aspects, such as redundant power supplies, i.e. such that each dual modular redundancy (DMR) channel is dependent on a different power supply. Yet, correct timing behavior as a technical safety aspect is out of scope of the reported platform. In [BRKS07] a number of generic redundancy mechanisms for LET based designs are mentioned. However, the authors strongly focus on an LET implementation based on TDM scheduling, which significantly limits the flexibility, as LET is interpreted rather as time triggering. Furthermore, the focus is

on code generation to develop for such platforms rather than the safety or fail-operational aspect of timing requirement. Other related work that is not domain specific can be categorized in two branches: First, system-level mechanisms to incorporate fault detection and recovery mechanisms such that timing guarantees can be provided and second timing analysis techniques that favour probabilistic bounds instead of static bounds. W.r.t. fault-tolerant operating systems we focus on two designs here: the ROMAIN framework [Dö14] and the COMPOSITE OS with extensions for fault tolerance [Par10]. In the two systems a different focus is made for the fault-tolerance aspect for real-time computing. The work in [SP15] and [SWP13] focuses on latent fault detection and a recovery with a real-time guarantee, i.e. that a timing bound on the recovery action is provided through analysis. The system reported on in [SP15] is a component-based system which tracks all communication between each component and the rest of the system (such as the event of component invocation, context switch or interrupt), and dynamically validates that execution behavior and timing conform to the models specified offline to analyze the system. Although the system can detect WCET overrun due to its inherent monitoring features, it suffers from the WCET dilemma laid out in Section 4.1, i.e. that conservative bounds on the WCET itself are necessary. However, the timing diversity approach in combination with it could prove effective, since in the timing diversity approach an exceedance of the (SL-LET) specification only requires to bring the timing channel in a defined state for the next execution, i.e. keeps the recovery overhead low while not requiring conservative WCET bounds. The only limitation is that an upper bound on the CET must be available for which the probability of a violation is sufficiently small – this can e.g. be achieved through extensive measurement campaigns.

The ROMAIN framework is a solution how the state of task replicas can be compared and treated on mismatch [AEDH12][Dö14]. In DMR mode recovery rolls back the replicas to their last valid state and re-executes them; in the Triple Modular Redundancy (TMR) mode, the state of the faulty replica is replaced with the state of a healthy replica. For the ROMAIN framework different scheduling variants have been proposed in [AQN+13], [Axe16], [RE17], and [Ram19]. All the reported scheduling variants are based on the BET model and provide timing bounds on the response-time even if recovery actions are necessary. In [AQN+13] a fork-join construct is proposed that divides a protected task in different stages and parallel segments of stages. Scheduling under partitioned SPP is investigated in [AQN+13] and later in [Axe16] revised due to optimism in the original approach. As in [Axe16] the priorities of task stages decrease as their deadlines increase, replicated tasks

perform worse when mapped in parallel than when mapped to the same core, i.e. if the redundancy is performed in time. This is due to the state comparison that cannot be handled well by the partitioned SPP scheme due to the lack of a performant synchronization. In an SL-LET based design, this problem can be overcome by SL-LET's implicit synchronization characteristic. How this issue can be overcome from a scheduling perspective is reported in [RE17] and [Ram19]. The authors apply gang scheduling, which is a co-scheduling variant that schedules groups of interacting tasks simultaneously. The "gangs" are the replicas that are mapped to SPP scheduled resources, however have to be mapped with the highest priority, which inevitably creates timing dependencies. These dependencies become a limitation if tasks of different criticalities are replicated and not enough resources are available to distribute them timing dependence free. In summary, both approaches provide timing bounds on the response-time of a diversified and redundant mapping, however they are not directly targeted at faults that cause a deviation of the response time. They rather catch faults that interfere with the correct function as such they could be used to harden timing diversity setups against such fault classes. Since also both scheduling variants suffer from the problem, that they require conservative WCET bounds (cf. Section 4.1), an SL-LET design with timing diversity based on optimistic worst-case bounds, resulting e.g. from measurements, can enhance the approach. The timing diversity approach can provide timing reliability although not necessarily conservative WCETs are used. Furthermore, timing diversity introduces a paradigm change in the design of real-time systems with such mechanisms, as it separates the design model from the implementation model. This allows the integration engineer to apply mechanisms such as [RE17] without having to go through the loop with the functional engineer to clarify timing aspects. The overarching SL-LET design provides clear deadlines (in the form of the LET write times) for applying the methods from related work during the integration.

The timing diversity approach also aims to relax the constraints on the conservativeness of the WCETs parameters, while maintaining a sufficient timing reliability. Probabilistic timing analysis (PTA) as an extension of the BET model is a research field of its own, which can coarsely be divided into the larger subfield of task WCET analysis and a smaller one that also aims at probabilistic response-time analysis. In [CKM+19] an extensive survey of past and contemporary probabilistic worst-case execution time (pWCET) analysis methods are presented. It impressively underlines why tight conservative WCET parameters are so hard to obtain on contemporary architectures and that a need exists to deal with this issue at the system

or cause-effect chain level. Within the field of pWCET analysis, often also referred to as just PTA, exists a line of work that again takes scheduling effects into account as a feedback on the observable pWCET behavior. In [KQA+13] and [DSA+13] specifically the cache evictions that a task may suffer because of preemption are addressed. However, employing such analysis techniques on a design is the ultimate counterpart to composable design, as it requires analyzing very specific configurations and compositions and this has to be done for every ever so small change. While [CKM+19] surveys pWCET analysis techniques, the authors of [BBB03] are one representative of work where pWCET bounds are used in the context of CPA, i.e. where probabilistic response time bounds are the goal. Similar to conventional performance analysis methods with conservative WCET bounds which compute latency metrics on data ([SMT+18], [FRNJ08]), pWCET based performance analysis also suffers from the problem that results are only intervals, i.e. predictability of data timing along a cause-effect chain is hard to obtain. A problem that is overcome by the SL-LET specified system, where a run-time ensures the adherence of the BET implementation to the SL-LET model.

5.3 Homogeneous Redundancy in SL-LET Designs

A generic method to handle stochastic error classes is to apply *homogeneous redundancy* for error handling, as e.g. proposed by ISO 26262 in [Int18b, Part 6 Table 5-1c]. The redundancy is referred to as *homogeneous*, as the two channels are identical and execute the same software, i.e. no diversification w.r.t. the implementation of the function is performed. The only limitation for this course of action is that run-time faults are no common cause faults, as e.g. a common power supply would be. Common cause faults would invalidate the statistical independence assumption from Eq. (5.1).

In addition to masking an execution time overshoot, homogeneous redundancy is also able to handle run-time faults due to unexpected hardware behavior, if the hardware is duplicated as well. In the following the concept of homogeneous redundancy with redundant hardware is applied to capture also this fault class and to ensure the statistical independence assumption for the timing behavior in both channels.

In principle DMR can be applied in two operation modes. Either to only detect that at least one channel has been struck by an error as a mismatch between channels, or switching to the correctly functioning channel. In the

latter case, the scheme requires, that the channel selection logic is capable of detecting the correctly functioning channel. Here, the latter variant is applied and the principle is depicted in Fig. 5.3. The setup features two redundant channels, and a channel selection mechanism. The channels consist of redundant computation in two SL-LET cause-effect chains $\mathcal{C}$ and $\mathcal{C}'$ as well as communication of their results to the channel selection

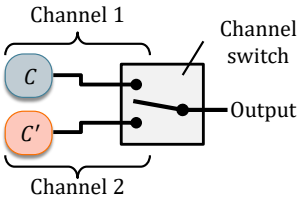


Figure 5.3: Generic homogeneous redundancy scheme with two redundant cause-effect chains C and C'

5.3.1 Timing Error Detection in SL-LET Designs

Obviously the channel selection mechanism cannot only be implemented by detecting the correctly functioning channel, but also by detecting the faulty channel. It is the nature of timing violations, that they can be detected by absence of an event at or before a point in time at which an event is expected. In an SL-LET design, the suitable events are the publication times of label instances, as the publication times are known in advance for each job producing them. Hence, it is possible to monitor whether these events are met for selecting the correctly working channel.

A SL-LET design whose implementation follows the BET execution semantic is free from error, i.e. behaves as specified, if the correspondence according to Definition 4.5.1 is fulfilled. Consequently, an error detection mechanism can be based on the requirements of Definition 4.5.1. The detection mechanism can e.g. be implemented in the SL-LET middleware that provides the correct data, i.e. the currently published, to a succeeding job. Here it has to be supervised that before publication time, the producing job has completed delivering data. Data over two label instances can remain identical, however, the middleware needs to be notified that the data is still “fresh”. If a job

violates event correspondence, and misses the time to publish new data, the error can be detected.

In a non-redundant setup, this detection mechanism still allows a fail-safe behavior, as the timing error is detected. Depending on the application, signaling the circumstance that a new label instance has not been produced can be used to trigger an appropriate failure handling strategy. For instance [MHMJZ20] asses methods how control software can still achieve stability in the presence of timing errors that result in lost or delayed data for the control algorithm.

5.3.2 Synthesis of DMR for LET Cause-Effect Chains

The goal is now to synthesise a structure that follows the DMR scheme from Fig. 5.3 from an effect chain specification in SL-LET. Fig. 5.4 shows in three steps, how a cause-effect chain $\mathcal{C} = (\lambda_1, ..., \lambda_n)$ can be protected by homogeneous redundancy. In the first step, the chain $\mathcal{C}$ is duplicated such that we now have $\mathcal{C}$ and $\mathcal{C}'$. In the second step, each chain is mapped. Depending on the assumptions for the error model, the mapping process has different constraints. If hardware characteristics are a fault class that trigger the error model or cause common cause errors, i.e. make errors statistically dependent, each chain must be mapped to a subset of the platform graph such that the mapping relations for $\mathcal{C}$ and $\mathcal{C}'$ map its tasks and labels to independent subsets of the platform graph $\mathcal{P}_{ch1}$ and $\mathcal{P}_{ch2}$:

$$\mathcal{P}_{ch1} \cap \mathcal{P}_{ch2} = \emptyset \tag{5.2}$$

In any case, sufficient independence between the possible errors in both chains must exist. Note that Eq. (5.2) assumes absence of any further common cause effects in the platform design. In a case where power glitches can have an impact on execution-time, e.g. redundant and independent power supplies are necessary to power the platform in order to avoid a common cause coupling. In the simplest case where a cause-effect chain only consists of a single task and ECUs are perfectly common cause error free under the assumed error model, the two steps only duplicate the task and map the redundant counterparts to separate ECUs.

The third DMR synthesis step consists of adding two interconnect tasks Φ_b and Φ_b' that deliver a copy of the labels $_n$ and $'_n$ from the ends of both cause-effect chains to the channel switch logic. For implementing the channel

switch logic, two options exist: This can either be done in the SL-LET run-time environment that takes care of label buffering, or by a receiving job itself. In the first case, only the label from the correctly working channel is published for reading by λ_c. More precisely, only the label that has arrived in time is published at time $D^{min}_{b,j} = D^{min}_{b'}$ for consumption by a job $\lambda_{c,k}$. In the second case, where a receiving job $\lambda_{c,k}$ itself checks the channels and performs the switch, the SL-LET run-time must allow that a reading job is able to determine whether it reads the label instance it is expecting from Φ_b and $\Phi^{'}_b$. For instance by checking a sequence number. This case is depicted in Fig. 5.4.

Similarly to the platform subsets for $\mathcal{C}$ and $\mathcal{C}^{'}$ in Eq. (5.2), the subsets for Φ_b and $\Phi^{'}_b$ must also be common cause error free. However, this requirement is more complicated to express than in Eq. (5.2), as the outputs of both channels must converge at the channel selection logic. Hence, the mapping relations ρ^{LET}_{Φ} for Φ_b and $\Phi^{'}_b$ inevitably have common elements at the convergence point. Typically, software is used at both ends to implement the copy action of Φ_b and $\Phi^{'}_b$, e.g. a communication stack on a joint resource. For *sufficient* independence between Φ_b and $\Phi^{'}_b$ it is therefore acceptable if elements from the realm of the channel switch logic's resources are shared, however the other resources Φ_b and $\Phi^{'}_b$ are mapped to must be disjoint: Formally, this means that the sets $\hat{\mathcal{R}}_{\Phi_b}$ and $\hat{\mathcal{R}}_{\Phi^{'}_b}$ consisting of all computation and communication resources Φ_b and $\Phi^{'}_b$ consume service on must be disjoint except for resources $\mathcal{P}_{sw}$ that are part of the channel selection logic:

$$\hat{\mathcal{R}}_{\Phi_b} \setminus \mathcal{P}_{sw} \cap \hat{\mathcal{R}}_{\Phi^{'}_b} \setminus \mathcal{P}_{sw} = \emptyset \tag{5.3}$$

With:

$$\hat{\mathcal{R}}_{\Phi_b} = \{r \mid (\Phi_b, r) \in \rho^{LET}_{\Phi}\} \qquad \forall\, r \in \hat{\mathcal{R}} \tag{5.4}$$

$$\hat{\mathcal{R}}_{\Phi^{'}_b} = \{r \mid (\Phi^{'}_b, r) \in \rho^{LET}_{\Phi}\} \qquad \forall\, r \in \hat{\mathcal{R}} \tag{5.5}$$

However, for *sufficient* independence between Φ_b and $\Phi^{'}_b$, the common resources must be free from common cause errors w.r.t. the assumed error model. Note that this must also hold for the implementation, more precisely

for common BET tasks in a BET implementation, e.g. if Φ_b and Φ_b' share the network stack on the receiving side.

The next section makes the argumentation why both channels only have a bounded dependence w.r.t. timing based on the timing dependency analysis from Chapter 2. First however, another important aspect of introducing DMR to an SL-LET design must be mentioned. While duplicating the cause-effect chain as shown does not alter the data age of either $_n$ nor $'_n$, the part of determining the correct channel and joining channels at the channel switch logic introduces a delay. A (direct) reader of $_n$ without the DMR involved, directly sees $_n$ in the data intervals $D_{n,i}$ of λ_n. Yet the consumer λ_c from Fig. 5.4 reads $_{n+1}$ or $'_{n+1}$, which is the copy of either $_n$ or $'_n$. Hence, the age of the data read by λ_c increases by the maximum read-to-write distance of Φ_b/Φ_b' – assuming that Φ_b and Φ_b' have identical periods and LETs. If

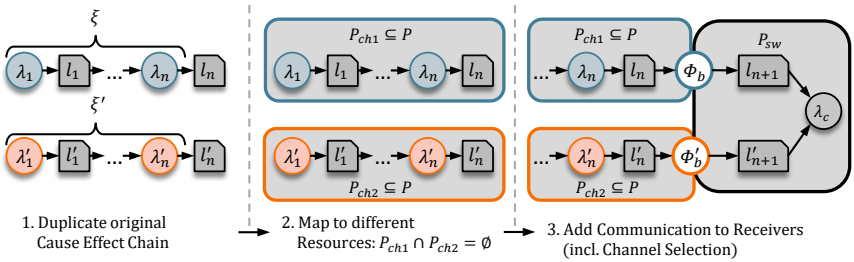


Figure 5.4: Steps to duplicate a cause-effect chain ξ for homogeneous redundancy

In order to be a meaningful design step, the DMR setup has to fulfill two properties. First, that the introduction of DMR has no adversarial effect on the design in terms of timing reliability and performance. While the check for performance is already part of the synthesis steps proposed, reliability is assessed in Section 5.3.4. Second, it has to be established that the timing behavior in the two channels is (sufficiently) independent according to the notions of Chapter 2. This is shown in the following.

5.3.3 Bounded Timing Dependence of DMR channels

In this section we will see that it can be shown with the methods from Chapter 2 that the channels of the DMR setup only have a bounded timing dependence. To perform a timing dependency analysis for a cause-effect chain that has been duplicated as described in the previous section, a BET implementation model is necessary. We therefore assume that for the SL-LET model resulting from the DMR synthesis process a corresponding BET model can be derived. The assumptions here are identical to the ones made for validation of an LET design in Section 4.6.2.

To recapitulate: every LET task λ_i has a corresponding BET task τ_{λ_i} or BET task chain. Model correspondence is ensured by an appropriate response-time requirement in the BET model. Furthermore, interconnect tasks Φ_j correspond to task chains with event precedences where the source $\tau_{\Phi\_j,Tx}$ and the sink $\tau_{\Phi\_j,Rx}$ of the chain are BET tasks on execution resources. The tasks $\tau_{\Phi\_j,Tx}$ and $\tau_{\Phi\_j,Rx}$ model the communication stack/middleware actions to handle network communication. Both tasks communicate sampling based with their preceding and succeeding tasks according to the cause-effect chain specified in the LET design model. Computation resources are assumed to be SPP scheduled and communication resources SPNP scheduled. Based on the resulting BET implementation model, the flow for the *bounded dependence* proof is shown on the conceptual example.

As TDGs tend to grow easily – at least for visual inspection – we first investigate a minimum example. The cause-effect chain w.l.g. is reduced to a single task $\mathcal{C} = (\lambda_1)$, that for longer chains produces the output label which is copied by the interconnect task to the channel switch logic in the redundancy setup. This reduced perspective of a single task is possible due to the injective property of data-flow (cf. Theorem 4.3.4) in LET cause-effect chains which results in deterministic input/output relations of label instances. In the resulting TDG, shown in Fig. 5.5, three connected components (indicated by blue boxes) can be identified. The first two describe the timing dependencies on the individual resources of the two channels. We can observe that the only connections (timing dependencies) from these components to the third component in the TDG originate from the output event models which become the input event models for $\tau_{\Phi\_j,Tx}$ and $\tau_{\Phi\_j,Rx}$. These are highlighted in red in Fig. 5.5. Consequently, bounded interference proofs are necessary for these four dependencies, as they resemble the only (timing) interference path from one channel to the channel switching. And consequently the only way how a timing failure in one channel chain propagate to the channel switch logic and impair it. Obviously, impairing correct timing behavior

of the switch has to be prevented for the DMR setup to be effective against timing errors in one channel.

Adding additional tasks on the resources of τ_{λ_i} and its redundant counterpart τ_{λ_i} does not change the general picture. For sufficient timing independence of the channels it is necessary, that the interference on any timing dependence path originating from either of the channels can be bounded. Obviously it is also required to ensure sufficient timing independence for the switch logic as well. To ensure this, a dependency analysis in connection with a confidence analysis or monitor synthesis can be used, as presented in Section 2.5 and Chapter 3.

It has to be noted, that the rationale for bounded timing dependence here is that the system is free of common cause failures[Int18b, Part 1, Clause 3.18] w.r.t. the timing in the two channels. A potential fault class which can result in common cause failures (CCFs) are common "components" (software and hardware) in both timing channels. Particularly, base software and identical hardware could cause dependencies between the channels. It will be elaborated later, how such dependence can be avoided, i.e. to achieve statistical independence of the timing failures which can be masked by the timing diversity setup. Applying measures to a design to avoid dependent failures [Int18b, Part 1, Clause 3.27] is a common task in safety engineering [Int18b, Part 9, Clause 7].

5.3.4 Reliability Considerations

In order to be able to evaluate whether a DMR setup actually improves the reliability of cause-effect chain timing, we can examine the design's success probabilities. We will use the common notion of reliability $\mathcal{R}(t)$ as a metric here. Generally speaking, $\mathcal{R}(t)$ denotes the probability, that a design is working without an LET violation in the time interval $[0, t]$. Formally expressed as:

$$\mathcal{R}(t) = P[\text{no LET violation in } [0; t]] \tag{5.6}$$

Reliability as a metric, broken down to individual tasks is also used e.g. in [SE09b] for formal analysis of CAN bus communication or [Axe16] for the formal analysis of BET task replicas on an multi-processor system-on-chip (MPSoC). The objective here is to derive the reliability of entire cause-effect chains and ultimately of the synthesised DMR setup. The knowledge of the

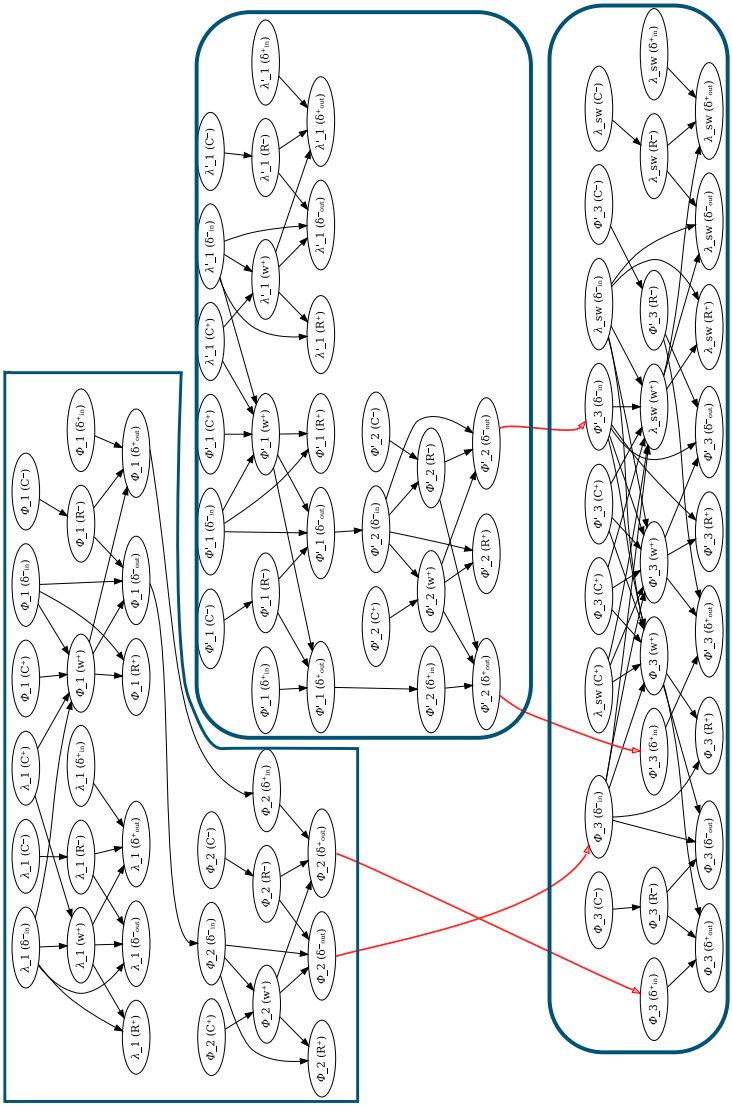


Figure 5.5: TDG (labeled with the idices of) of the corresponding BET model showing that the only timing dependence between the DMR channels and the channel switch are the output event models from the two channels, assuming that an upper bound on transmitted data exists.

reliability function for the entire design, sometimes also referred to the survival function, can also be used to calculate further common metrics to asses dependable systems, such as the mean-time to failure (MTTF):

$$\text{MTTF} = \int_0^\infty \mathcal{R}(t)\, dt \tag{5.7}$$

It has already been mentioned that the DMR setup for timing diversity is able to handle two error classes: First, timing errors that result in LET violations, and second stochastic hardware errors that cause the failure of a platform element. However, the latter is a byproduct of the mapping to disjoint platform elements. Before considering these concrete error models, we focus on the DMR setup's reliability in general. To compute $\mathcal{R}(t)$ for entire cause-effect chains we first focus on the reliability of timely execution of individual tasks.

Definition 5.3.1: Success

The event such that the BET execution of the j-th job of an LET task λ_i meets the LET LET_i is referred to as $S_{i,j}$.

Definition 5.3.2: Success Probability

The *success probability* $P[S_{i,j}]$ is the probability that the BET execution of the j-th job of an LET task λ_i meets LET_i.

Hence, we can conclude, that a cause-effect chain instance $\mathcal{C}_{k,l}$ is successful if all of its jobs $\lambda_{i,j} \in \mathcal{C}_{k,l}$ yield success according to Definition 5.3.1. Deriving the probability of this, however, is not trivial, since it requires knowing in which time interval the jobs of $\mathcal{C}_{k,l}$ are active. However, we can derive the reliability $\mathcal{R}_i(t)$ of any task λ_i, as the probability that all jobs $\lambda_{i,1}, ..., \lambda_{i,n}$, which have completed with success (cf. Definition 5.3.1) and have been released in the time interval $[0; t]$ by:

$$\mathcal{R}_i(t) = P[S_{i,1} \wedge S_{i,2} \wedge ... \wedge S_{i,n}] \quad \forall \lambda_{i,j} : 0 \leq \hat{R}_{i,j} \tag{5.8}$$

This is done under the assumption that errors are statistically independent w.r.t. individual jobs. Based on this assumption, it can be further reasoned

about the reliability of a cause-effect chain instance $\mathcal{C}_{i,j}$ (cf. Definition 4.3.3), which is successful if all jobs belonging to it are successful:

$$\mathcal{R}^{\mathcal{C}}_{i,j} = P[\bigwedge_{a,b\,:\,\lambda_{a,b}\in\mathcal{C}_{i,j}} S_{a,b}] \tag{5.9}$$

Hence we can define the reliability of a cause-effect chain in the interval $[0;t]$:

Definition 5.3.3: Cause-Effect Chain Reliability

A (serial, branch free) cause-effect chain $\mathcal{C}_i = (\lambda_1, ..., \lambda_n)$, is successful, if all instances of it started in the interval $[0;t]$ are successful. Its reliability is hence given by:

$$\mathcal{R}^{\mathcal{C}}_i(t) = \prod_{j\,:\,\mathcal{C}_{i,j}\text{starts in }[0;t]} \mathcal{R}^{\mathcal{C}}_{i,j}$$

Theorem 5.3.4: Cause-Effect Chain Reliability Bound

For a serial, brach free, cause-effect chain $\mathcal{C}_i = (\lambda_1, ..., \lambda_n)$, the reliability in the time interval $[0;t]$ can be bounded by the reliability of the tasks contained in it, i.e. when these yield success according to Definition 5.3.1:

$$\begin{aligned}\mathcal{R}^{\mathcal{C}}_i(t) &= \prod_{j\,:\,\mathcal{C}_{i,j}\ \text{starts in }[0;t]} \mathcal{R}^{\mathcal{C}}_{i,j} \\ &\geq \prod_{k\,:\,\lambda_k\,\in\,\mathcal{C}_i} \mathcal{R}_k(t)\end{aligned}$$

Proof 5.3.5:
The success probability of individual task instances are considered independet of each other. Since $\mathcal{R}_a(t)$ for a task λ_a considers all jobs of a task λ_a started in $[0;t]$ it is a lower bound for a subset of jobs, as $\mathcal{R}_a(t)$ conservatively considers all (independent) jobs. Hence, for a cause-effect chain all potential jobs are considered, not just the jobs contained in the cause-effect chain instances as given by Theorem 4.3.4.

□

Note that this is a conservative assumption which considers jobs that might not necessarily have to be considered. This is for instance the case when $\mathcal{C}_i$ contains tasks with different release periods and offsets. In this case, oversampling and undersampling occurs, and hence not every job of a task $\lambda_j \in \mathcal{C}_i$ is part of any cause-effect chain instance. In this case a job $\lambda_{j,k}$ may not be successful, however all cause-effect chain instances are. Nevertheless, we consider this case as an unsuccessful execution. Hence, Theorem 5.3.4 is a conservative approximation of the reliability.

The SL-LET DMR setup that we consider consists of two parallel cause-effect chains out of which only one has to complete successfully, i.e. timely. Furthermore, also the channel switch logic has to work successfully, i.e. determine which channel has completed according to the LET scheme. Using the theory of reliability block diagrams (RBDs) the success probability of such complex structures can be computed [Mos58]. The theory assumes that a system consists of a network of serial and parallel strands of blocks, where a single block can be made up from a nested network of blocks. Through computation rules for serial and parallel block strands the reliability for the entire network can be computed.

We consider the three essential parts of the SL-LET DMR system (channels
i
r
t

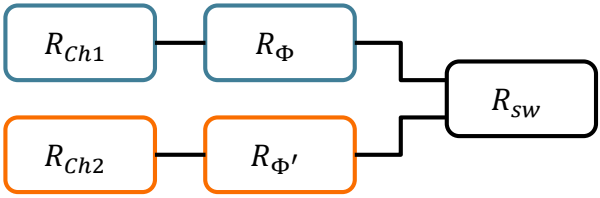


Figure 5.6: Reliability Block Diagram (RBD) to calculate the success probability of an SL-LET DMR setup

Note that blocks in RBDs can be nested, i.e. a block can again contain an entire reliability network. Hence, we assume w.l.g. that the $\mathcal{R}_{Ch1}(t)$ and $\mathcal{R}_{Ch2}(t)$ blocks abstracts the initial cause-effect chain and the blocks $\mathcal{R}_{\Phi}(t)$ and $\mathcal{R}_{\Phi'}(t)$ abstract the communication towards the channel switch logic.

The overall reliability of the blue ($\mathcal{R}_1(t)$) and orange blocks ($\mathcal{R}_2(t)$) in series can be computed by:

$$\mathcal{R}_1(t) = \mathcal{R}_{Ch1}(t) \cdot \mathcal{R}_{\Phi}(t) \tag{5.10}$$
$$\mathcal{R}_2(t) = \mathcal{R}_{Ch2}(t) \cdot \mathcal{R}_{\Phi'}(t) \tag{5.11}$$

The parallel strands are successful if either one channel or both channels are successful. The complementary event is both strands fail:

$$P[\text{both fail in } [0;t]] = (1 - \mathcal{R}_1(t))(1 - \mathcal{R}_2(t)) \tag{5.12}$$

Hence, the success probability of parallel strands ($\mathcal{R}_{\parallel}(t)$) can be computed as one minus the complementary event, resulting in a success probability for the parallel strands of:

$$\mathcal{R}_{\parallel}(t) = 1 - (1 - \mathcal{R}_1(t))(1 - \mathcal{R}_2(t)) \tag{5.13}$$

The overall reliability is again a series concatenation of $\mathcal{R}_{\parallel}(t)$ and the $\mathcal{R}_{sw}(t)$ block: This yields a DMR success probability of:

$$\mathcal{R}_{DMR}(t) = \mathcal{R}_{\parallel}(t) \cdot \mathcal{R}_{sw}(t) \tag{5.14}$$

This result can already be used to reason when the DMR setup is meaningful, i.e. improves system reliability:

$$\mathcal{R}_{\parallel}(t) \cdot \mathcal{R}_{sw}(t) \overset{!}{\geq} \mathcal{R}_{\text{non-redundant}}(t) \tag{5.15}$$

Whereas the reliability of the non-redundant variant is the reliability of for the non duplicated LET cause-effect chain according to Theorem 5.3.4. The logical constraint when to apply DMR is that the redundant variant is more reliable than the non-redundant one.

Now we turn to the particular fault type we envision to mitigate with *timing diversity*: violations of execution time assumptions that would ultimately lead to response-time requirement violations. While the previous reliability estimation does consider stochastic faults, which also have their error effect at runtime, the faults considered here conceptually already happen at design-time, but the effects only become observable at runtime. Hence, they are also

at-runtime errors. With design-time faults the success in the sense of the reliability estimation is that expected behavior as encoded by the CL-model is actually observable at runtime. The reliability is thus the probability that the observable behavior is covered by the model expectations.

The model for design-time faults that is taken as the basis here makes two assumptions: First, that the *structure of the design is correct*. W.r.t. the model layers of the CLM this means that the graphs that form the basis of a model layer are structurally correct, i.e. that a design fault does not introduce or omit edges or nodes in the CLM. Second, it assumes that the modeled errors, i.e. different runtime behavior than described by the model, originate from *wrongfully specified parameters* as the design fault. Wrongfully specified parameters can have two types of effects on the verification and analyses performed on the model: First, an erroneous specification causes overly conservative verification and analysis results. As all the (timing) verification methods considered in this thesis are conservative, these design-faults only have benign effects at runtime. The consequence is that this type of faults can be seen as immediately masked as no hazardous effects can occur at runtime. Although it is undesirable that system resources are poorly utilized it does not cause hazards at runtime and thus they are not considered. Second, a faulty parameter specification can lead to non-conservative behavior at run-time. This effect is targeted by the assumed error model for timing diversity. The essence of it being that parameters which contain a fault in their specification can cause run-time failures, such as exceeding a response-time bound and eventually the violation of an LET leading to an invalid model correspondence.

Furthermore, the error model is limited to the assumption that specification faults do not trigger as common cause errors in both DMR channels simultaneously at runtime. At first sight, this seems counter-intuitive as the software in both channels might only be duplicate instances and hence the design time fault would be the root cause making both channels dependent. However, this common case only appears if execution in both channels occurs in a lockstep manner, where the timing behavior of CPU microarchitecture is deterministic, and scheduling and input data is synchronized. Particularly, scheduling also includes the remaining load of the system and not only the software under investigation. Instead, contemporary embedded high-performance architectures as e.g. the Renesas RCar H3 SoC [Ren] or Intel's scalable Denverton platform [Int20] exhibit a different picture. For computation performance, the RCar H3 SoCs rely on ARM Cortex-57 [ARM13] cores. Like comparable CPU microarchitectures these cores feature out-of-order and speculative execution features. While traditional safety ar-

chitectures as e.g. the Infineon TriCore [Inf12] refrain from such techniques or restrict speculation to data reads, commercial off-the-shelf (COTS) CPUs that are also used in smartphones, etc., however, rely on these techniques for optimal average case performance [ARM13]. This produces inherent non-determinism w.r.t. the timing behavior of even a single instruction in a core's pipeline.

Further, performance architectures run a complex operating system stack. Virtualization allows to execute a number of virtual machines (VMs) with their own OS on a single SoC. This indirection w.r.t. hardware access further increases non-determinism of the execution time behavior of tasks residing in one VM. As (efficient) virtualization requires more complex memory access handling through MMUs that support virtualization, data access latencies become dependent on the memory hierarchy and how it is accessed. While caches and cache coherence have a significant influence [LGR+16], also page faults or TLB misses can trigger lengthy recover procedures. Particularly the fact "who" else and at which point in time access memory becomes crucial. Static WCET estimation techniques as assumed by classical system-level timing analysis like CPA or MPA are unable to follow the pace of technical innovations for memory and cache hierarchies [WEE+08]. The newer field of probabilistic timing analysis [CKM+19] is also limited by the fact that the software under analysis is not in full control of the platform. Furthermore, a significant number of future applications functionally require rich OSes such as Linux. For instance applications built on robot operating system (ROS) and data distribution service (DDS). Alike, efforts in recent years were made to decrease the timing uncertainty of the PREEMPT_RT Linux kernel [RMF19] [OO16] [OCOC20]. While the research for PREEMPT_RT does not allow to provide conservative estimates, especially due to the reasons already named above, it allows reasonable bounds for the "common worst-case" and the knowledge that a true worst-case rarely appears. In conclusion, the view embraced here for the fault model accepts inherent weaknesses of contemporary WCET estimation techniques and how operating systems and computational platform specifics influence WCET behavior. In addition to this, DMR introduces a method into the design that remedies this weakness. Under the assumption that all sources of timing uncertainty lead to different timing traces in both channels the common cause fault of violating a specified parameter can be excluded. This assumption is backed by the discussed timing uncertainties. The independence can even be increased if the design makes sure that the background load on a resource or shared SoC has a different execution trace characteristic in both channels. The consequence of differing execution traces in both DMR channels is that

no common cause fault event happens. A manifesting error (at runtime) hence only affects *one* channel and consequently the SL-LET based DMR setup protects against the timing error, as it can select the channel which is on time.

The timing in the erroneous channel violates the assumptions for model correspondence of the BET execution model and the SL-LET design model. Hence, selecting the properly working channel can be performed by monitoring the LET of the interconnect tasks that deliver the data to the channel switch logic.

The overall timing reliability of the setup can also be evaluated with the RBD methodology applied in Fig. 5.6 with three basic steps:

- Assignment of a reliability to each LET task that models that the LET will be met
- Computing the reliability of the two cause-effect chain strands that form a DMR channel as a serial connection of reliability blocks
- Computing the reliability of the entire DMR setup as a serial connection of the LET task that implements the channel switch logic and the two parallel strands that form the two channels.

Note that the channel switch logic does not necessarily have to be implemented in a separate LET task, but can be part of a middleware implementation. In this case the reliability that this implementation provides the correct data *on time* to the subsequent subscriber is used.

As concrete error probabilities are hard to obtain discretization of error probabilities as done by safety standards and confidence analysis could be applied. However, in the remainder of this work, the main focus is put on the fact whether sufficient (statistical) independence between the timing channels can be established, such that timing diversity increases the overall reliability . Following Eq. (5.15) it can be seen that the DMR setup improves reliability if sufficient independence for common cause errors can be argued for a particular design. Consequently, the DMR setup can improve timing reliability for designs where truly conservative specification is impossible and e.g. only tracing-based WCET estimation is possible. The setup helps to mitigate non-conservative specification (design faults) that can eventually result in runtime timing errors.

5.4 Experiments and Case Studies

5.4.1 Timing Diversity Case Study: Environment Perception

As laid out, a particular side effect of the timing diversity concept is that the demands on the specification of WCETs can be relaxed, even if the timing of the original cause-effect chain itself is already critical. This is due to the fact that the redundancy can mask the effects of a timing errors. Hence, this case study presents the design of a complex automated driving cause-effect chain, that in contrast to the lateral controller example from Chapter 2 through Chapter 3 can not be easily implemented on timing predictable architectures, or architectures that by configuration can achieve a timing predictable behavior, such as for instance the Aurix TriCore Family [Har13]

Figure 5.7: 3D perception cause-effect chain from autoware.auto as ROS nodes

The example cause-effect chain that we consider here is a 3D-perception application taken from autoware.auto [Aut] and shown in Fig. 5.7. In this chain, LIDAR images from a Velodyne laser scanner obtained at 10 Hz are processed by a point cloud filter before a ray-ground filter determines the ground plane in the resulting data. For prototyping a recorded actual image sequence is fed from a packet capture (pcap) file to the Velodyne driver. The non-ground points in the resulting data are subsequently processed by an euclidean cluster algorithm, clustering points into objects. The output of this step could now for instance be used for sensor fusion, to fuse it with objects detected by additional sensors or to check the objects for plausibility. For the sensor fusion time coherence of the data is essential, which requires some form of synchronization. At minimum the data age of the objects must be known to resolve objects with high spacial accuracy in the environment of the vehicle.

In terms of integrating the algorithms on a platform, they are characterized by the fact that they are computationally demanding and thus require high performance platforms. Besides this, they often entail the need for rich operating systems that are at least POSIX capable, such as Linux or the AUTOSAR adaptive platform.

For the case-study we choose an Intel x86_x64 platform which already can be found in automotive OEMs' roadmaps for perception systems, e.g. the Intel Denverton platform [Yos20]. The case study itself is the 3D perception stack of the Autoware.auto project [1], which is based on ROS2 [2] and runs on Linux as operating system. ROS2 is a data-centric middleware essential built around the DDS protocol suite. In ROS2 the producer and consumer of data are essentially ROS nodes, whereas each of the nodes can exist as a separate binary. The ROS2 API abstracts the DDS API of several DDS vendors for the ROS client library. Essentially, the DDS specification would allow SL-LET to be implemented as part of its interface, i.e. to become an SL-LET middleware, as DDS takes care of publishing and delivering data.

The purpose of this case study is to show, that a cause-effect chain like the one of the autoware.auto 3D perception example can be modeled and specified with SL-LET and that the timing diversity scheme can be applied to it in order to increase its timing reliability despite the fact that no hard WCET bounds are available. Here, we focus on the three computationally expensive nodes, namely the point cloud filter, the ray ground classifier and the euclidean cluster node.

As a first step, either the execution time and activation pattern characteristics, or directly the latency characteristics of the three nodes must be obtained. Note that in the former case, latency characteristics would be obtained through analysis of execution time and activation patterns of software on the platform. Since for the experiments platforms no execution time analysis tools are available, we resort to measuring the latency by instrumentation – if executed at the highest priority the latency effectively becomes the execution time. Obviously this neglects blocking effects but is a reasonable first approximation. For this purpose callbacks that record the start and end time of a when a node is processing an image are inserted. Hence, the time between recorded start and end time can be treated as a job. The ROS middleware and nodes of the autoware.auto example used here run on Linux. The Linux kernel in principle is a static priority scheduler that distinguishes different policies. While the "normal" scheduling policies execute on priority level 0 (lowest), the SCHED_FIFO policy that is used for the ROS nodes here, assigns priorities between 1 (low) and 99 (highest). The Linux documentation refers to SCHED_FIFO as a real-time policy [SCH]. In the SCHED_FIFO policy, a thread that becomes runnable and has a higher priority than the currently executing one will always preempt any currently

[1] autoware.auto
[2] ros2.org

running thread of lower priority. Threads on the same priority are executed in FIFO order. For the measurements the PREEMPT_RT kernel extensions are applied. Measurement-based execution times are also used in the evaluation and case study of [CBLB19] who proposes a response-time analysis for ROS2 nodes under the SCHED_DEADLINE policy of the Linux kernel.

To demonstrate the complexity and the numerous side effects of modern platforms and Linux as an operating system, the measurements for each node are performed in four different settings. To generate the four settings we use a combination of two configurations with two alternating options. The first configuration option is whether background load is applied to the system, and second which power saving strategy is used. As background load, we resort to *mprime*, which searches for prime numbers with special properties [MR]. Mprime is chosen since the load it generates is not I/O limited and mainly generates stress on the CPUs. Power saving as a feature is particularly interesting, since it is somewhat inevitable on modern performance architectures but can cause counter-intuitive effects as can be seen in the measurements. It also demonstrates that a lot of effects can not be treated in isolation but all its dependencies are also hard to consider due to their variety. In the concrete case, power saving is disabled by choosing the *performance governor* of the Linux kernel instead of the default power management strategy which is the *ondemand governor*.

5.4.1.1 Execution Time Measurement Results

Fig. 5.8 shows the response time measurements obtained for the ray ground estimation node while processing sensor data from a $\approx$ 8 min test drive with new images sampled at $\approx$ 10 Hz. For this measurement, the node received the highest priority in the system, i.e. the measured response-time is effectively an execution-time measurement. For the remaining nodes, the kernel's *ksoftirqd* which manages soft interrupts interferes as it was put on the second highest priority. The latter is necessary to ensure smooth replay of the pcap file with sensor data. The remaining ROS nodes are assigned a priority of the SCHED_FIFO policy. No other threads are put on a priority from one to 99, i.e. other load is scheduled under the regular policies. Specifically the background load that is applied in two experiments. The results are shown as violin plots, which show the relative frequency of different execution time/latency values. The plots include a marker for the median of the data and a box indicating the interquartile range, as in standard box plots.

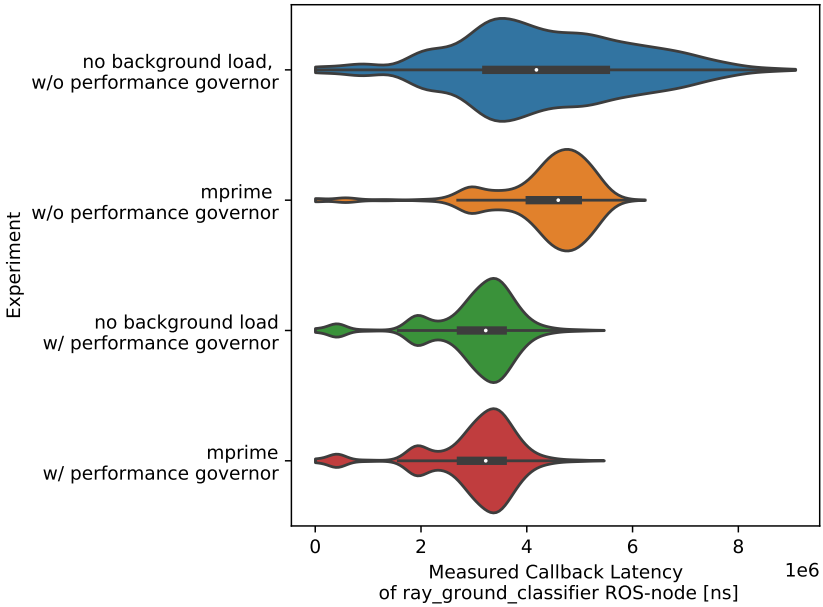


Figure 5.8: Relative frequency distribution of response-times of the ray ground classifier node

What can be observed in Fig. 5.8 from the first two experiments (orange and blue violins), is initially counter-intuitive. The frequency distribution of response times is more compact if background load is applied to the platform, whereas it spreads if only the ROS chain is executed. Intuitively, one would assume the narrow relative frequency distribution of response times / execution times if the software executes in isolation without interference. In these experiments the default power management strategy of the Linux kernel is used and the effect can be explained by the way the default power management strategy works: It decreases the number of cycles, in which the kernel reduces the frequency of the core, based on the *entire* system load. Hence, a higher loaded system generates shorter response times, as the cores run with the maximum frequency which is thermally possible. It is important to note, that the thermal budget of modern CPUs is typically the limiting factor here, as continuous operation at maximum performance is usually outside the long-term sustainable operational envelope. Due to this reason power management is inevitable. Even the performance governor used in two experiments here, can not guarantee that all cores will always run at the highest frequencies, since the hardware itself dictates the maximum possible speeds based on the thermal budget. The kernel's governor is hence limited by these values. However, we can observe that in experiments with the performance governor enabled (green and red violins), the frequency distribution of response times is similar, disregarding whether background load is applied to the system or not. In the demonstrated case, no further influence on the thermal budget is present – at least not intentionally. However, this might be different in real automotive conditions, where e.g. mounting positions and long term mechanical stress on heat conductive material can influence the thermal budget of cores. This is an easily overlooked aspect of modern performance architectures.

What can be initially concluded from these experiments is that for individual tasks a wide spread of the execution time / latency can be observed. The frequency distribution is altered with a slightly different configuration of the hardware and software platform, including the other load a system executes. It can also be observed in Fig. 5.8, that all four experiments do not exhibit significant outliers, and that although median values and observed maxima vary an estimated WCET / WCRT including a safety margin could be derived from the measurements. Such a specification can be subsequently used for an SL-LET specified and BET implemented system with timing diversity in which remaining timing errors can be masked due to timing diversity. However, this does not yet show that a timing diversity scheme with redundant channels can sufficiently handle the situation – this is shown

in the last part of Section 5.4.1.3 for the SL-LET based timing diversity setup from the next section.

Due to different core affinities, measurements of the latency between the start callback and the end callback for the other two ROS nodes can also be

Figure 5.9: Sequential execution of the nodes in one chain instance

Particularly, the relative frequency distribution of response-times of the point cloud filter in Fig. 5.10 shows the same "anomaly" as for the ray ground classifier in the experiment with the default power management (blue and orange violins). However, the relative frequency of response-times of the euclidean clustering algorithm in Fig. 5.11 for the experiment with the default power saving strategy appear as expected, i.e. with a longer tail and higher median value for the experiment with background load (orange violin) compared to no background load (blue violin). This effect cannot be

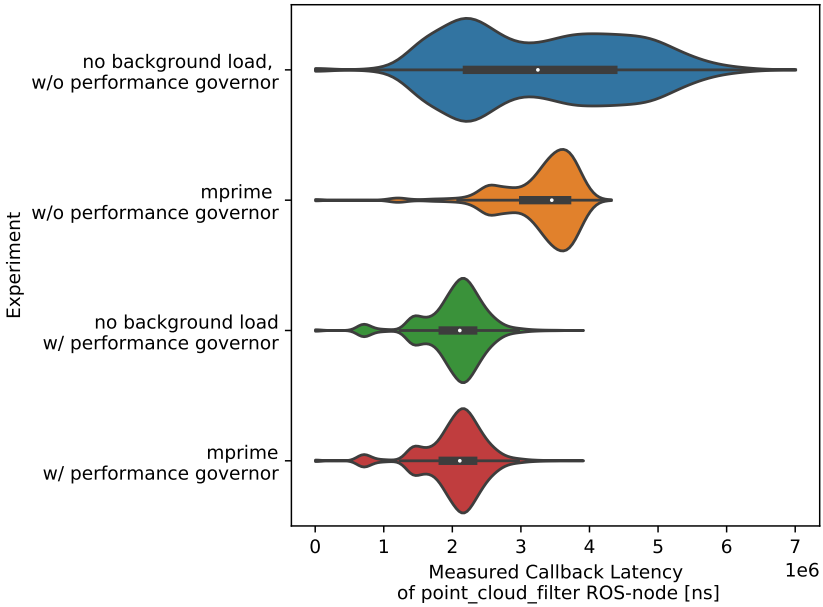


Figure 5.10: Relative frequency distribution of response-times of the point cloud filter node

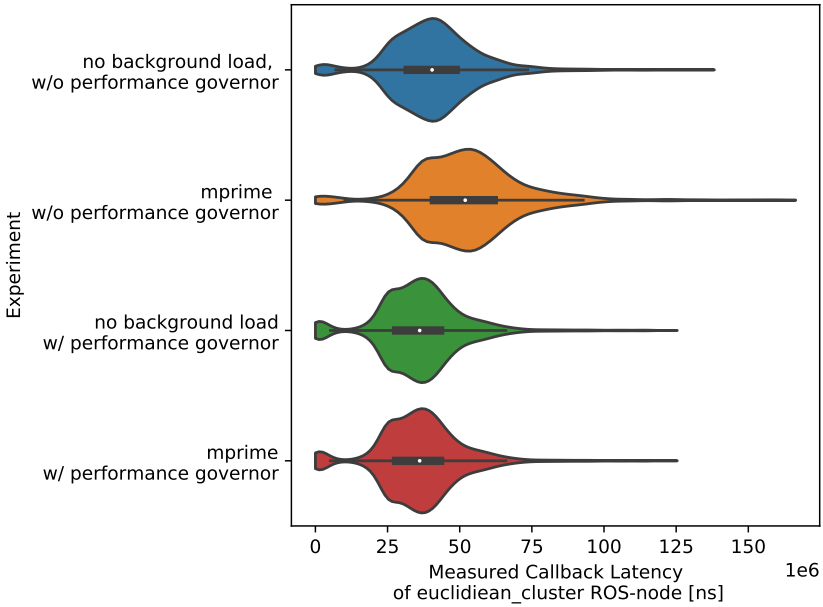


Figure 5.11: Relative frequency distribution of response-times of the euclidean cluster node

explained with the impact of the ondemand power governor. Nevertheless, response times observed in Fig. 5.10 and Fig. 5.11 allow to specify a WCRT for BET execution and to apply the timing diversity pattern to catch residual timing errors, as all observed response time samples are in the same order of magnitude.

5.4.1.2 Specifying the Chain in SL-LET

Based on the measured execution times/latencies from Fig. 5.10, Fig. 5.8, and Fig. 5.11, LETs are specified in Table 5.1. The chain is triggered with a period of 100 ms corresponding to the 10 Hz of the new image arrival from the LIDAR's pcap file.

Table 5.1: SL-LET Specification for the system in Fig. 5.7 partitioned into a cause-effect chain $\mathcal{C} = \{\lambda_1, \lambda_2, \lambda_3\}$.

Task	Description	LET	Period	Offset
λ_1	point cloud filter	5.7 ms	100 ms	0 ms
λ_2	ray ground filter	8 ms	100 ms	LET_1
λ_3	euclidean cluster	120 ms	100 ms	$LET_1 + LET_3$

Table 5.1 shows one possible SL-LET specification for the cause-effect chain presented in Fig. 5.7. By aligning the tasks with offsets to the LET of the predecessor, the chain achieves a minimum end-to-end response time compared to the case where all tasks operate with the same period but without offsets. It has to be noted that $\mathcal{C}$ here exhibits a constant data-age, as all tasks $\lambda_i \in \mathcal{C}$ execute with the same period. As in the case study in Table 5.1, the tasks are aligned front-to-back, the end-to-end read-to-write latency (identical to the data-age here) is the sum over the LETs, i.e. $\mathrm{R2W}_{\mathcal{C}}^{-} = \mathrm{R2W}_{\mathcal{C}}^{+} = 133.7\,\mathrm{ms}$. The end-to-end data age will also stay identical when the cause-effect chain is ported to a different platform, as long as the LETs of all $\lambda_i \in \mathcal{C}$ are still met under normal operation. Executed with the timing diversity scheme and timing errors, the chain $\mathcal{C}$ will still yield the same functional results (given identical input data) compared to non-diversified execution, as long as at least one channel is free from timing errors for each chain instance. However, the LET of the interconnect task that is needed due to the diversity scheme has to be added. Yet, this latency can be abstracted in the order of a

few ms, e.g. for Ethernet-based communication, and thus keeps the resulting end-to-end latency reasonable [TE16a] [ATED14] [TAE15] [TSAE16b].

The overall performance of the cause-effect chain in terms of read-to-write latency is largely dependent on the LETs. While timing diversity can compensate LETs errors, the fact how aggressively they can be shortened is a matter of the desired timing reliability. Naturally, this requires further investigations for specific combinations of applications (algorithms), platforms, and configuration of platforms, as they determine whether the required diversity is in the execution and response time profile of a task. The values chosen here for illustration, are rather aggressive. While the LETs safely and with additional margin conform to the measurements with either background load and/or the performance governor they are only above the 50%-quartile in the case of the standard power governor (blue violin in Fig. 5.10, Fig. 5.8, and Fig. 5.11).

5.4.1.3 Catching SL-LET Deadline Violations with Timing Diversity

To check whether the proposed DMR approach for timing diversity can be effective we need to obtain measurements that allow to compare two jobs/instances of all the LET tasks. These two jobs need to process the same data, as they do in a DMR setup. Therefore, recorded traces with identical input data are aligned and the latency of both executions compared. The diversity in this setup is introduced through the fact that the kernel's soft interrupt handling interferes with node execution, no core pinning is done (hence other workload in the system is able to influence the cache behavior), and disabling the performance governor which resulted in the largest spread of latencies. Due to the unpredictability introduced by these factors, it can be expected that the timing channels behave differently.

A result's data point consists of the latencies obtained for corresponding node executions in both channels. Further each value is annotated in which channel it was obtained. From the individual latencies, the difference in latency is computed. The results are shown in a scatter plot for which Fig. 5.12 provides the explanation how to read the scatter plot. A scatter plot depicts each result as a single point. In the plot, points are categorized by which channel resulted in the maximum latency of a pair, i.e. categories are either 'Channel 1' or 'Channel 2'. The y-axis provides the value of the maximum of the two measured latencies. The x-axis shows the difference of the latencies of both jobs processing the same data. Note that theoretically a third category is necessary for points with an x-value of 0 ns, as in this case the latency in

both channels is equal. However, in the experiment results in Fig. 5.13 such a case was not observed. To interpret the data points, Fig. 5.12 shows the deadline, i.e. the LET, for each node as a red line. For points below this line both jobs meet the LET. The blue line represents $f(x) = x + deadline$. Since, the x values are the latency difference of a redundant job pair, the orange area between the red and the blue line contain all pairs, where the job from the indicated category is above the deadline but the other one is below the deadline. Consequently, points in this area mark job pairs, where the timing diversity scheme actively masks the LET violation, regardless of the cause of the LET violation. Only points above the blue line can not be handled with the timing diversity approach, as an LET violation in both channels would occur.

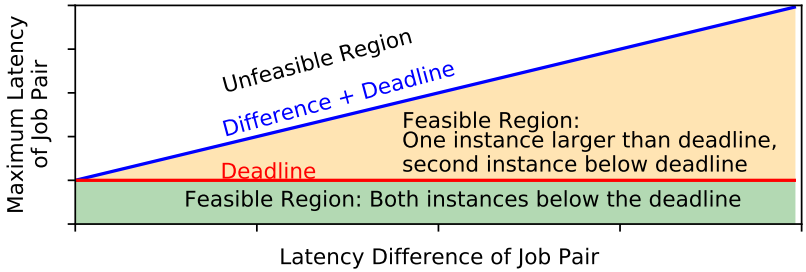


Figure 5.12: Explanation of the scatter plot evaluating the timing diversity experiment

To show the general applicability of the timing diversity concept to complex cause-effect chains, we perform the experiment with the ondemand governor in both DMR channels. The results are depicted in Fig. 5.13 for all three ROS nodes, respectively LET tasks. The red line in all three (sub)plots, marks the specified LET from Table 5.1.

Interpretation: First, we can observe in Fig. 5.13, that no instances are in the unfeasible region, i.e. where the timing diversity scheme is unable to handle LET errors. However, such instances would appear if the deadline / LET is lowered further. Similarly, increasing it would push the region between the red and the blue line upward, i.e. fewer cases would have to be caught by timing diversity. Yet, this would also result in longer end-to-end latencies over the SL-LET chain. Hence, the values for LETs have to be carefully chosen in relation to the cause-effect chain and hardware / software platform at hand. Second and most important, it can be observed in Fig. 5.13, that the points

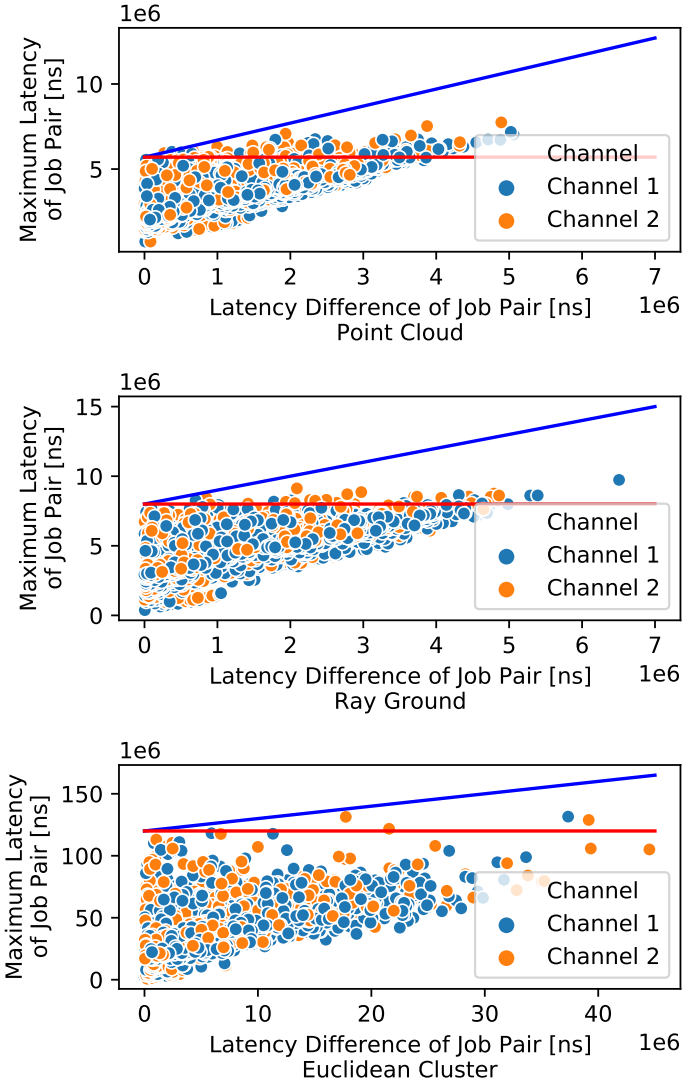


Figure 5.13: Scatter Plots showing the maximum latency of a job pair over the difference in latency for the pair as explained by Fig. 5.12

in the area where the DMR setup can correct LET violations are a mixture from both categories, i.e. channels. This is an indication that the DMR setup generally works, as not always the same channel has to be corrected and that there is reason to assume that there is some level of statistical independence that should be investigated in future work. Furthermore, since no points are above the blue exclusion line, where no correction is possible, the experiment also supports the claim that the DMR setup increases the overall timing reliability. This is the case, since all the events in the area between the red and the blue line would result in a timing failure, i.e. violation of the LET, if the setup is not duplicated. In conclusion, for the parametrization of the 3D perception chain in Table 5.1 together with the LIDAR data of an eight-minute test drive [1], the SL-LET DMR setup demonstrates its suitability. For this parametrization the timing diversity scheme catches all LET violations observed in the experiment, i.e. for a particular test drive / stream of LIDAR data, whereas a non-redundant variant would have failed multiple times (cf. Fig. 5.13).

5.4.1.4 Statistical Test of the Independence Hypothesis

In the context of student's master thesis the statistical independence assumption for response times was investigated further. Results thereof are summarized and submitted in [MHCE21]. In this work, the environment perception chain from Fig. 5.7 was tested with the same LIDAR data as above, but on a different hardware platform, which is depicted in Fig. 5.14. The two performance hosts thereby implemented the two timing channels. The processors of the two hosts have six-cores without simultaneous multi-threading (SMT). Their configuration is similar to the experiment above but differs in details, like Linux Kernel version, and concrete priority assignment. The Kernel version employed here is 5.6.19-rt12 with PREEMPT_RT. All the nodes of the ROS chain receive priorities in ascending order, whereby the node that performs the replay of the pcap file receives the lowest priority in the chain and the driver node, which translates the pcap data into LIDAR images, receives the second lowest. The point cloud filter and the ray ground classifier node are pinned to core 0 at priorities 46 and 47, respectively, while the euclidean cluster node is pinned to core 1 at priority 48. Power-saving options, like C-States, are disabled and the *performance governor* is enabled. All cores ran at a fixed clock frequency of 3.5 GHz. This configuration, at first glance, reduces the diversity in the setup. This is due to the fact that

[1] https://autoware-auto.s3.us-east-2.amazonaws.com/route_small_loop_rw.pcap accessed:15.11.2020

the experiment targeted caches as the source of the diversity. Therefore, both hosts were stressed with background load from the stress-ng tool suite [Kin21] to foster diversity.

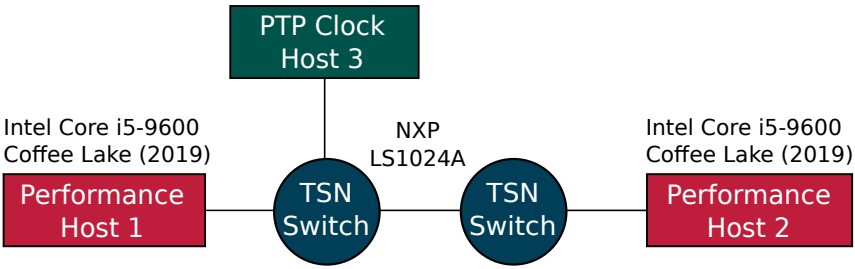


Figure 5.14: Platform used to obtain measurements for the statistical test. Source: Robin Hapka.

On each host, all remaining four cores, i.e. cores 2 to 5, execute the cache stress test from stress-ng, which performs random widespread memory read and writes to thrash the cache. This workload not only stresses the cache alone but also causes the core to be fully loaded. All stress-ng threads are executed on a user-level priority.

Since the hardware platform is different from the one in the previous experiment, and LETs where previously chosen aggressively, the LETs for the nodes are adapted to be equally aggressive as above. They are based on initially measured latencies, i.e. the approach is the same as above. The task parametrization is provided in Table 5.2.

Table 5.2: Alternative SL-LET Specification for the system in Fig. 5.7 on the platform from Fig. 5.14.

Task	**Description**	**LET**	**Period**	**Offset**
λ_1	point cloud filter	1.3 ms	100 ms	0 ms
λ_2	ray ground filter	2.7 ms	100 ms	LET_1
λ_3	euclidean cluster	68 ms	100 ms	$LET_1 + LET_3$

The latency measurements, were performed 56 times with the same LIDAR data, i.e. 112 times if executions in the timing channels are counted individ-

ually. By feeding in the same LIDAR data on each run of the chain, the node jobs become comparable, as each n-th job of a node processes the same data. The response times are categorized into two categories: No LET violation and LET violation. More precisely, the first category ranges from 0 ms to the respective LET for each node (cp. Table 5.2), and the second category from the LET to infinity. The resulting contingency table has two dimensions, i.e. the both timing channels, with two categories each.

Since due to the long run-time of the experiments, only 56 comparable repetitions are available. Therefore, Fisher's exact test[1] was applied to test the statistics. It is particularly effective for $2x2$ contingency tables and small random sample sizes. Both is the case here.

Fisher's exact test's null-Hypothesis H_0 is that the data is independent in the two dimensions of the contingency table are statistically independent. In the case here, the two dimensions are the timing channels, sub-categorized in whether a job has completed below the LET or above the LET. The test's alternative hypothesis H_1 is that the data is dependent. We choose a confidence level of 95%. Fisher's exact test was performed for all the 4715 samples processed by the ROS nodes in the 56 repetitions, i.e. each node performs 4715 computations for each run of the pcap file. Due to effects of the automatic lttng tracing, the execution of the first/last sample may not be evaluated correctly – these obvious measurement errors were omitted in the evaluation. The results are listed in Table 5.3.

Table 5.3: Results of Fishers' test on the response-time data from both channels of each sample

	Point Cloud (λ_1)	**Ray Ground** (λ_2)	**Euclidean Cluster** (λ_3)
Statistically significant **Independence**	4708	4709	4710
Statistically significant **Dependence**	6	3	0

[1] SciPy 1.5.4: https://docs.scipy.org/doc/scipy-1.5.4/reference/generated/scipy.stats.fisher_exact.html

For, in total only 9 samples, statistical significant independence could not be shown, and the alternative hypothesis was adopted by the test. The decision parameter of Fisher's exact test is the computed probability of obtaining a distribution at least as extreme as the one that was actually observed, assuming that the null hypothesis is true. At a significance level of $\alpha = 0.05$, this probability must be at least 0.05 or higher. In six out of the nine cases – twice for the ray ground nodes and four times for the point cloud node – the probability is only ≈ 0.035, the remaining three cases range between 0.011 and 0.017. However, for the overwhelming majority of tested samples, Fisher's exact test shows a statistical significant independence of the response times in the two timing channels, i.e. on the two hosts. In these cases, the probability result approaches ≈ 1.

Interpretation: There are multiple factors that may influence the result of the statistic test: First, there is the random sample size, which is fairly small with only 56 samples. Although Fisher's test is much better suited for this situation than e.g. the χ^2 test, it could cause the nine results where the alternative hypothesis was adopted. Second, the tests are performed on categorized data. Since the data is categorized based on a basically arbitrary deadline, this choice obviously has an influence, as a change of the (specified) deadline also changes the contingency tables. The choice of a suitable deadline is highly application specific, hence it has to be taken into account when the timing diversity scheme is considered for improving the timing reliability of a design. Third, the algorithms in the processing chain of this case study have a dependence on the input data. Hence, it has to be studied whether the size and run-time complexity caused by the input data has an influence on the response time behavior. Such a dependence could be interpreted as a common cause failure which could not be captured by a DMR scheme such as timing diversity.

5.4.2 Case Study: Exploiting Timing Diversity for ASIL Decomposition

In this case study we take another step back to the point of specifying timing requirements and how they related to high level functional safety goals. The question that shall be answered here is how the timing diversity can be applied such that the safety and integration process benefits from it. We again consider an environment perception function for which we assume the chain in Fig. 5.7 is one possible implementation, or at least a part of it.

Assume that in the course of safety engineering one of the safety goals for the sub function of the laser scanner specifies as: [1]

> *An object in the detection area of the laser scanner must be in the object map no later than x time units.*

In the safety process this safety goal receives an ASIL value. This safety goal is imposed on the implementation of the cause-effect chain implementing the LIDAR-based object detection. Effectively, this means for an SL-LET design that a maximum read-to-write distance $\mathrm{R2W}_C^+$ from reading the sensor output to writing the object map, must be guaranteed with the assigned ASIL.

ISO 26262 in Part 9 Clause 5 explicitly embraces the concept of *ASIL decomposition*. It is a concept of "apportioning of redundant safety requirements to elements, with sufficient independence, conducing to the same safety goal, with the objective of reducing the ASIL of the redundant safety requirements that are allocated to the corresponding elements" [Int18b, Part 1 Clause 3.6]. The intent behind this concept is that if in the design of the architecture is sufficiently independent and redundant elements exist, then it is possible to allocate a specific safety requirement to two (or more) of these elements. In this case, the standard allows that the redundant requirements may receive a lower ASIL than its original parent. As a consequence, this can significantly reduce the development and documentation process efforts if parts of a design element can be attributed with a lower ASIL. Nevertheless, the decomposed requirement is effectively the same requirement. The possible decompositions w.r.t. ISO 26262 are provided in Clause 5.4.9 of Part 9 in [Int18b] and are depicted in Fig. 5.15.

Note that the original ASIL before the decomposition is applied is given in parentheses – this is required by the standard in order to be able to trace where a requirement came from [Int18b, Part 9, Clause 5.4.8]. Decompositions can also be applied recursively, however, for each decomposition it must be argued that *sufficient independence* between the redundant elements exists [Int18b, Part 9, Clause 5.4.10]. Furthermore, if the decomposition results in a function plus a safety mechanism, the safety mechanism must carry

[1] Note that this case study is limited in exhaustiveness and only focuses on aspects relevant here. Further the argumentation brought forward here is largely based on ISO26262:2018 which does not yet cover automated vehicle functions in general such as ISO/PAS 21448:2019 does. However, since the case study is not focused on risk and hazard identification and classification which is the major focus in the ISO/PAS 21448:2019 initiative but rather on the integration, the steps discussed here are applicable anyway.

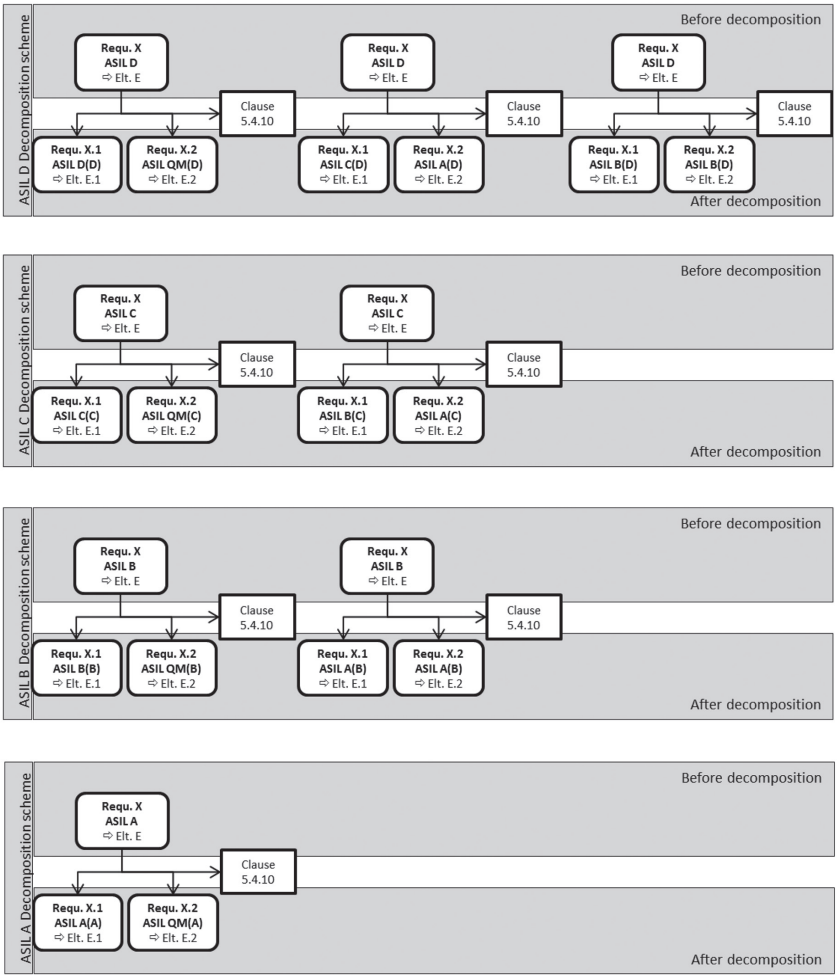


Figure 5.15: Possible Decompositions of ASILs (Source: [Int18b, Part 9, Clause 5.4.8, Figure 2])

the higher ASIL. E.g. in the case of QM(D) + ASIL-D(D) the safety mechanism carries the ASIL-D(D) [Int18b, Part 9, Clause 5.4.7].

We can observe in Fig. 5.15 that symmetric decompositions (e.g. ASIL-D = B(D) + B(D)) and asymmetric decompositions (ASIL-D = QM(D) + D(D)) are possible. For both types we show how timing diversity can be applied:

5.4.2.1 Symmetric decomposition

W.r.t. timing diversity the symmetric decompositions can be applied to the timing behavior of the chain in a timing diversity setup directly. In this case both channels receive identical requirements on the timing behavior in the channels, e.g. B(D) if an ASIL-D is decomposed. Obviously, establishing timing guarantees with a confidence corresponding to an ASIL-B is much easier to accomplish than for ASIL-D, which substantially reduces development cost and effort. Furthermore, it relaxes the sufficient independence argumentation w.r.t. other software on the platform resources a timing channel is implemented on, as only assigning the decomposed value it does not taint the entire remaining software's timing behavior by its higher ASIL [Int18b, Part 4, Clause 7.4.2.2]. Furthermore, it is either possible to confidence requirements minimal with the assignment method from Section 3.1, or to ensure them with monitors as described in Section 3.2. Note that the monitors in this case operate on the BET implementation, ensuring the adherence to the SL-LET specification only and do not directly enable the safety mechanism as such.

In the symmetric decomposition the channel switch of the timing diversity scheme is a common element for which [Int18b, Part 9, Clause 5.4.10] requires the original ASIL requirement as it can not be made sufficiently independent of the timing behavior. However, the objective to be carried out by the channel switch is relatively easy, as it only has to determine if either channel is on time at pre-designated points in time. Hence, the higher, original and non-decomposed, ASIL is comparatively easy to achieve for the channel selection in respect to the timing behavior of the complex function. *Sufficient independence* of the timing behavior between the two channels can be shown in the form of bounded dependence as described in Section 5.3.3.

In conclusion, the symmetric decomposition allows relaxing timing requirements on complex software components and their underlying hardware platform to easier achievable levels. In doing so it maintains a *fail-operational* characteristic, w.r.t. timing errors, as the timing diversity setup is able to

successfully mask them with the assurance of the original safety requirement. Furthermore, the setup also catches "babbling idiots" in one of the channels. These can easily be detected at the channel switch logic, as the logic also kind of works as a monitor, i.e. it only forwards or publishes data at the point in time that is given through the SL-LET specification. Hence, the "babbling idiot" can be easily identified and silenced.

5.4.2.2 Asymmetric decomposition

Asymmetric decompositions require extending the argumentation for the safety goal. The argumentation is extended such that either the object must be detected after x units, or the detection chain must signal an error within the same time frame. Signaling the error, i.e. that no processing of the lidar image has happened, has the purpose that other functions that use the resulting object maps can work with this information. A possible scenario is for instance that a plausibility check of object data from another technology, e.g. radar, based on the lidar's object map can be skipped for this image. The safety mechanism in this case obviously is the functionality to detect the timing error and signal this circumstance to consumers of the object map. Hence, in these decompositions, this functionality must receive the higher ASIL [Int18b, Part 9, Clause 5.4.7]. The timing behavior itself receives the lower ASIL, which depending on the original ASIL are: QM(D), C(D), QM(C), A(C), QM(B), QM(A). In the QM(x)-cases, ensuring the timing is hence reduced to a pure quality management effort. Nevertheless, the timing is important, as it avoids a potential functional degradation. Hence, to avoid these quality impairments timing diversity can still be applied, with the benefit that its implementation and integration is not subject to the rigid requirements of the safety standard w.r.t. the process, etc. [Int18b, Part 6].

In conclusion, the asymmetric decomposition follows a fail-safe pattern w.r.t the safety design. However, the quality management effort in the form of timing diversity can significantly improve product quality as detachment of the function is prevented through high timing error resilience due to the diversity scheme. In these cases the LETs for applications might be tailored more aggressively as the margin between expected response-time and LET is not required for safety purposes but is only a quality management effort.

Chapter 6

Conclusion

This thesis is motivated by the fact that timing as an *extra-functional* property can have an impact on the functional safety properties of a system. This influence of timing manifests in design as well as synthesis aspects of systems, which spawned the three initial research questions. In short these questions were:

1. How can hidden timing dependencies be detected?
2. Which automated timing engineering efforts can ease the burden of designers?
3. How can different levels of timing safety requirements be integrated in mixed-critical systems?

This thesis has shown in Chapter 2 how a model-based development flow supports the systematic identification of timing dependencies, based on a conservative approach, as it assumes dependence by default. Only if the analysis can show independence or at-worst bounded dependence based on the confidence ratings of parameters that go into quantified timing analysis methods, a design is cleared – meaning that freedom from (timing) interference can be assumed. The approach presented in Chapter 2 showed the importance of traceability of requirements over multiple model layers. If this is not given, there are problems to identify destructive interference that violates the freedom from interference paradigm in a complex design with multiple functions hosted in the same system. Such complex systems, however, are at the forefront of what is currently discussed for next-generation

vehicular systems. The traceability of safety requirements on which the timing behavior of the implementation has an influence was also shown in Chapter 2. The approach presented there, goes together with new timing analysis techniques that have recently been shown for systems with complex call structures [SE17].

Chapter 3 has presented answers to questions two and three. The chapter showed: First, how based on the top-down principle of the safety process, timing parameter confidence values can be allocated to all timing parameters a safety-critical timing property depends on. Second, how the automatic synthesis of a monitoring network can isolate different levels of criticality from each other. Particularly, the aspect that both levels of criticality are isolated up to their required safety level, which quantifies the risk of experiencing a timing failure. In the direction from a higher to lower level we can safely assume that timing failures of the higher criticality are rarer it has higher confidence requirements towards its parameters. As a consequence a non-common-cause failure influencing the higher criticality component is sufficiently unlikely for the lower critical one – hence freedom from interference w.r.t. to the different levels is reasonably established. In the direction from the lower to the higher confidence level, monitors as "confidence boosters" together with the quantified timing analysis guarantee the freedom from interference property. The method proposed in Chapter 3 showed feasibility in the sense that for randomly generated systems which resemble a zone like architecture reasonable amounts of monitors are necessary.

But as already concluded in Chapter 4 all the engineering tweaks introduced here do not solve the challenges raised in the research questions satisfactorily, as they are unable to tame the timing unpredictability that arises with the interaction of software with contemporary multicore platforms with shared caches, MMUs, etc.. The high confidence into timing parameters needed for mutual isolation of higher safety levels can simply not be guaranteed by the methods available. Even if analysis techniques that would provide reasonable confidence into timing parameters, it is safe to assume that they in turn would be fairly complex. Consequently, the complexity of the design itself would only be pushed to the next "lower level" – not limiting the effort for designers in general. Hence, the research questions where extended as to how the unpredictability of such platforms can be made sufficiently predictable for timing requirements which are technical safety requirements.

To reduce the complexity of the design in general, a shift in the design paradigm was ultimately necessary. With SL-LET such a design and pro-

gramming paradigm was presented. The paradigm allows composable design of end-to-end timing properties along cause-effect chains. A fact that is unprecedented with the contemporary BET timing model as design and implementation model. Furthermore, the timing composability allows specifying predictable timing of data. A fact that we identified as very valuable especially for future applications such as perception and automated driving applications. SL-LET alone can improve timing error detection in complex cause-effect chains, as it defines concrete points in time at which a result or data must be available *by design* and *transparent* to the function developer. In the state-of-the-art BET design assumed in Chapter 2 and Chapter 3 such points are only implicitly defined in the process of monitor configuration. A process most nontransparent for the function engineer, as it is hidden by the complexity of the timing model. Based on SL-LET each execution of a software block within an LET task, the function software can be notified with low overhead, whether the data it reads is the instance it expects. This allows to place timing error handling into the function software itself where necessary, paving the way for advanced control-software error handling strategies as presented in [MHMJZ20]. With BET execution and monitoring, the timing exception handling remains in the realm of the timing engineer who might be agnostic of the exact impact of the violation of the extra-functional property. In any case she/he is the wrong person to handle the exception from the functional perspective.

Building on the property that timing violations are easily detectable in SL-LET, Chapter 5 has introduced the concept of *timing diversity* to tackle the issue of the unpredictable timing by the fact that abnormally high [1] execution times are stochastically a rare event. Based on the observation that different software structures lead to different execution time behavior, this thesis could show that the timing diversity approach increases timing reliability, i.e. the probability of an execution being on time, improves under it for both fault classes: stochastic hardware faults that lead to prolonged execution times and design faults where the assumed worst-case was too optimistic. While the statistical independence assumption is subject to further research, e.g. how it can be substantiated based on platform properties or software composition, the foundation of the timing diversity concept are provided in Chapter 5. Especially the applicability of the concept to ASIL tailoring suggests that timing diversity is able to reduce costs although redundant execution is necessary for it, as with lower (tailored) ASILs the process cost are far less. Obviously the break-even point between these types of cost

[1] In the sense that the execution time is extremely higher than the median in a long-tailed distribution.

(investment vs. recurring costs per piece) has to be decided on concrete projects.

Since it is proposed to implement SL-LET by following a BET execution semantic with a suitable middleware that handles the LETs label communication at the right point in time, the timing dependency analysis technique as well as monitoring are not worthless. It is fact that the timing safety property in SL-LET resides with fulfilling the LET specification and safety mechanisms are also controlled base on the LETs. But, maintaining a low rate of timing exceptions, i.e. violations of LETs is a quality management effort. This effort can easily be supported by the timing dependency analysis method presented in Chapter 2 that automatically reveals other BET tasks on which the implementation of an LET task depends on. Sorting out which dependencies are acceptable for a suitably low rate of timing exceptions can now be taken care of by the real-time engineer without having to deal with the burden of planing or coordinating the instantiation of a safety mechanism. With SL-LET and traceable timing safety requirements this can be easily addressed in the software design as the software designers are provided with a timing composable interface.

Bibliography

[AAD17] AADL: Architecture analysis & design language. https://www.sae.org/standards/content/as5506c/, 2004 - 2017.

[AEDH12] Philip Axer, Rolf Ernst, Björn Döbel, and Hermann Härtig. *Designing an analyzable and resilient embedded operating system.* Gesellschaft für Informatik e.V., 2012. Accepted: 2018-11-06T10:58:22Z ISSN: 1617-5468. URL: http://dl.gi.de/handle/20.500.12116/17855.

[Aer09] Aeronautical Radio Inc. *ARINC Specification 664P7-1: Aircraft DataNetwork, Part 7 – Avionics Full Duplex Switched Ethernet (AFDX) Network*, september-2009 edition, September 2009.

[AHE16] L. Ahrendts, Z. A. H. Hammadeh, and R. Ernst. Guarantees for runnable entities with heterogeneous real-time requirements. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1646–1651, March 2016.

[ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, March 2004. doi:10.1109/TDSC.2004.2.

[AQN+13] P. Axer, S. Quinton, M. Neukirchner, R. Ernst, B. Dobel, and H. Hartig. Response-Time Analysis of Parallel Fork-Join Workloads with Real-Time Constraints. In *2013 25th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 215–224, July 2013. doi:10.1109/ECRTS.2013.31.

[ARI19] ARINC. *Avionics Application Software Standard Interface, Part 0, Overview of ARINC 653*, 2 edition, August 2019.

[ARM13] ARM Ltd. ARM® Cortex®-A57 MPCore Processor Technical Reference Manual. page 575, 2013.

[Ass13] EAST-ADL Association. EAST-ADL Domain Model Specification. Technical Report V2.1.12, EAST-ADL Association, November 2013.

[ATED14] Philip Axer, Daniel Thiele, Rolf Ernst, and Jonas Diemer. Exploiting shaper context to improve performance bounds of Ethernet AVB networks. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014. ISSN: 0738-100X. doi:10.1145/2593069.2593136.

[Aut] Autowarefoundation. 3D perception stack. URL: https://autowarefoundation.gitlab.io/autoware.auto/AutowareAuto/perception-stack.html [cited 2020-11-25].

[AUT19a] AUTOSAR. *Specification of Timing Extensions, Release 19-11*, 2019. Release 19-11.

[AUT19b] AUTOSAR GbR. *AUTOSAR - Specification of Operating System*, R19-11 edition, November 2019.

[AUT20] AUTOSAR. AUTOSAR Meta-Model (MMOD), Release 20-11. https://www.autosar.org/fileadmin/user_upload/standards/foundation/20-11/AUTOSAR_MMOD_MetaModel.zip, 2020. Release 20-11.

[Axe16] Philip Axer. *Performance of Time-Critical Embedded Systems under the Influence of Errors and Error Handling Protocols*. Cuvillier, Göttingen, February 2016.

[BB18] Harald Bucher and Jürgen Becker. Electric Circuit- and Wiring Harness-Aware Behavioral Simulation of Model-Based E/E-Architectures at System Level. *2018 IEEE International Systems Engineering Symposium (ISSE)*, 2018. doi:10.1109/SYSENG.2018.8544434.

[BBB03] Alan Burns, Guillem Bernat, and Ian Broster. A Probabilistic Framework for Schedulability Analysis. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Rajeev Alur, and Insup Lee, editors, *Embedded Software*, volume 2855, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. Series Title: Lecture Notes in Computer Science. URL: http://link.springer.com/10.1007/978-3-540-45212-6_1, doi:10.1007/978-3-540-45212-6_1.

[BBK19] Harald Bucher, Jürgen Becker, and Simon Kamm. Cross-layer behavioral modeling and simulation of E/E-Architectures using preevision and Ptolemy II. In *Proceedings of the 2019 Summer Simulation Conference*, SummerSim '19, pages 1–12, Berlin, Germany, July 2019. Society for Computer Simulation International.

[BCC+17] A. Bucaioni, A. Cicchetti, F. Ciccozzi, S. Mubeen, and M. Sjödin. A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation From EAST-ADL. *IEEE Access*, 5:9005–9020, 2017. Conference Name: IEEE Access. doi:10.1109/ACCESS.2016.2641218.

[BD19] Alan Burns and Robert I. Davis. Mixed Criticality Systems - A Review. Technical Report 12th Edition, Department of Computer Science,University of York, York, UK, March 2019.

[BDM+16] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. Synthesizing job-level dependencies for automotive multi-rate effect chains. In *Int. Conf. Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2016.

[BDM+17] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture*, 80, 2017. doi:https://doi.org/10.1016/j.sysarc.2017.09.004.

[BDM+18] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. Analyzing End-to-end Delays in Automotive Systems at Various Levels of Timing Information. *SIGBED Rev.*, 14(4):8–13, January 2018. URL: http://doi.acm.org/10.1145/3177803.3177805, doi:10.1145/3177803.3177805.

[BDT10] Matthias Biehl, Chen DeJiu, and Martin Törngren. Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems. In *Proceedings of the ACM SIGPLAN/SIGBED 2010 Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES '10, pages 125–132, New York, NY, USA, 2010. ACM. URL: http://doi.acm.org/10.1145/1755888.1755907, doi:10.1145/1755888.1755907.

[BE18] Matthias Beckert and Rolf Ernst. The IDA LET Machine - An efficient and streamlined open source implementation of the

Logical Execution Time Paradigm. In *International Workshop on New Platforms for Future Cars (NPCar at DATE 2018)*, March 2018.

[Bec20] Matthias Beckert. *Scheduling Mechanisms for Efficient and Safe Automotive Systems Integration.* PhD thesis, Universitätsbibliothek Braunschweig, January 2020. URL: https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00067003, doi:10.24355/DBBS.084-201911070747-5.

[Ber15] Peter Johannes Bergmiller. *Towards functional safety in drive-by-wire vehicles.* Springer, 2015. doi:10.1007/978-3-319-17485-3.

[BG03] S. K. Baruah and J. Goossens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Transactions on Computers*, 52(7):966–970, July 2003. Conference Name: IEEE Transactions on Computers. doi:10.1109/TC.2003.1214344.

[BME16] Matthias Beckert, Mischa Möstl, and Rolf Ernst. Zero-time communication for automotive multi-core systems under SPP scheduling. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–9, September 2016. doi:10.1109/ETFA.2016.7733563.

[BNB07] Enrico Bini, Marco Di Natale, and Giorgio Buttazzo. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems*, 39(1-3):5–30, April 2007.

[BRKS07] Christian Buckl, Matthias Regensburger, Alois Knoll, and Gerhard Schrott. Generic Fault-Tolerance Mechanisms Using the Concept of Logical Execution Time. In *13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007)*, pages 3–10, December 2007. doi:10.1109/PRDC.2007.14.

[BSBA14] Klaus Becker, Bernhard Schatz, Christian Buckl, and Michael Armbruster. Deployment Calculation and Analysis for a Fail-Operational Automotive Platform. *arXiv:1404.7763 [cs]*, April 2014. arXiv: 1404.7763. URL: http://arxiv.org/abs/1404.7763.

[BYRLB18] Chris Becker, L. Yount, S. Rosen-Levy, and J. Brewer. Functional Safety Assessment of an Automated Lane Centering System (Report No. DOT HS 812 573). page 129, August 2018. doi:(ReportNo.DOTHS812573).

[CBLB19] Daniel Casini, Tobias Blaß, Ingo Lütkebohle, and Björn B. Brandenburg. Response-Time Analysis of ROS 2 Processing Chains Under Reservation-Based Scheduling. In Sophie Quinton, editor, *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:23, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 1868-8969. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10743, doi:10.4230/LIPIcs.ECRTS.2019.6.

[CDF+14] Carlo LaTorre, DeJiu Chen, Frank Hagl, Henrik Lönn, Luis P Azevedo, Mark – Oliver Reiser, Martin Walker, Renato Librino, Sandra Torchiaro, Sara Tucci-Piergiovanni, and Tahir Naseer Qureshi. MAENAD Deliverable D6.1.3 - Case study analysis and safety assessment. Deliverable D6.1.3 3.1, February 2014. URL: https://www.maenad.eu/public/Deliverables/MAENAD_Deliverable_D6.1.3_V3.1.pdf.

[Cer14] Certification Authorities Software Team (CAST). Position Paper CAST-32 Multi-core Processors, May 2014. URL: http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/ [cited 2015-04-06].

[CKM+19] Francisco J. Cazorla, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella, and Tullio Vardanega. Probabilistic Worst-Case Timing Analysis: Taxonomy and Comprehensive Survey. *ACM Computing Surveys*, 52(1):14:1–14:35, February 2019. doi:10.1145/3301283.

[Con68] Melvin E. Conway. How do committees invent? *Datamation*, April 1968. URL: http://www.melconway.com/research/committees.html.

[DAF+19] Alexander Dörflinger, Mark Albers, Björn Fiethe, Harald Michalik, Mischa Möstl, Johannes Schlatow, and Rolf Ernst. Demonstrating Controlled Change for Autonomous Space Vehicles. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2019.

[DBBL07] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, April 2007.

[Die16] Jonas Diemer. *Predictable Architecture and Performance Analysis for General-Purpose Networks-on-Chip*. Verlag Dr. Hut, 2016. OCLC: 951636740.

[DSA+13] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of Probabilistic Cache Related Pre-emption Delays. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 168–179, July 2013. ISSN: 2377-5998. doi:10.1109/ECRTS.2013.27.

[Dö14] Björn Döbel. *Operating system support for redundant multithreading*. PhD thesis, TU Dresden, Dresden, Deutschland, 2014. URL: http://os.inf.tu-dresden.de/papers_ps/doebel-phd.pdf.

[EBTLR12] Rolf Ernst, Alan Burns, Lothar Thiele, and Jimmy Le Rhun. Mixed critical system design and analysis. In *Proceedings of the tenth ACM international conference on Embedded software*, EMSOFT '12, pages 247–248, New York, NY, USA, 2012. ACM. URL: http://doi.acm.org/10.1145/2380356.2380401, doi:10.1145/2380356.2380401.

[EGA13] Arbeitskreis EGAS. Standardisiertes E-Gas Überwachungskonzept für Benzin und Diesel Motorsteuerungen. Technical Report Version 5.5, Arbeitskreis EGAS, July 2013. URL: https://www.autotec.ch/technik/pdf/ms_E-GAS_Ueberwachung.pdf.

[EJL+03] J. Eker, J. W. Janneck, E. A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, January 2003. Conference Name: Proceedings of the IEEE. doi:10.1109/JPROC.2002.805829.

[EK72] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, April 1972. URL: http://doi.acm.org/10.1145/321694.321699, doi:10.1145/321694.321699.

[EKG18] Rolf Ernst, Leonie Köhler, and Kai-Björn Gemlau. System Level LET: Mastering Cause-Effect Chains in Distributed Systems. In *44th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, October 2018.

[EN16] R. Ernst and M. Di Natale. Mixed Criticality Systems - A History of Misconceptions? *IEEE Design Test*, 33(5):65–74, October 2016.

[FAH14] Thomas Frese, Hans-Jörg Aryus, and D. Hatebur. Safety analysis as backbone for the oem safety process – systematic support for concept- and system development including v&b activities. In *Praxisforum Fehlerbaumanalyse 2014*, 2014.

[FGH06] Peter H. Feiler, David P. Gluch, and John J. Hudak. The Architecture Analysis & Design Language (AADL): An Introduction:. Technical report, Defense Technical Information Center, Carnegie Mellon University, Fort Belvoir, VA, February 2006. URL: http://www.dtic.mil/docs/citations/ADA455842, doi:10.21236/ADA455842.

[FRNJ08] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS) at the 30th Real-time Systems Symposium (RTSS)*, 2008.

[GAM+15] Giovani Gracioli, Ahmed Alhammad, Renato Mancuso, Antônio Augusto Fröhlich, and Rodolfo Pellizzoni. A Survey on Cache Management Mechanisms for Real-Time Embedded Systems. *ACM Computing Surveys*, 48(2):32:1–32:36, November 2015. doi:10.1145/2830555.

[GHPP18] Danlu Guo, Mohamed Hassan, Rodolfo Pellizzoni, and Hiren Patel. A Comparative Study of Predictable DRAM Controllers. *ACM Transactions on Embedded Computing Systems*, 17(2):53:1–53:23, February 2018. doi:10.1145/3158208.

[GKEQ21] Kai-Björn Gemlau, Leonie KÖHLER, Rolf Ernst, and Sophie Quinton. System-level Logical Execution Time: Augmenting the Logical Execution Time Paradigm for Distributed Real-time Automotive Software. *ACM Transactions on Cyber-Physical Systems*, 5(2):14:1–14:27, January 2021. doi:10.1145/3381847.

[GM02] J. C. Geffroy and Gilles Motet. *Design of Dependable Computing Systems*. Springer Netherlands, 2002. URL: https://www.springer.com/de/book/9781402004377, doi:10.1007/978-94-015-9884-2.

[GSME17] Kai-Björn Gemlau, Johannes Schlatow, Mischa Möstl, and Rolf Ernst. Compositional analysis for the waters industrial challenge 2017. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Dubrovnik, Croatia, June 2017.

[HAE17] Robin Hofmann, Leonie Ahrendts, and Rolf Ernst. CPA – Compositional Performance Analysis. In Soonhoi Ha and Jürgen Teich, editors, *Handbook of Hardware/Software Codesign*, pages 1–31. Springer Netherlands, Dordrecht, 2017. doi:10.1007/978-94-017-7358-4_24-2.

[Ham19] Zain Hammadeh. Deadline Miss Models for Temporarily Overloaded Systems. September 2019. ISBN: 9781678285708. URL: https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00066886, doi:10.24355/dbbs.084-201909020857-0.

[Har13] Jens Harnisch. Predictable hardware: The AURIX Microcontroller Family. In *Workshop on Worst-Case Execution Time Analysis (WCET)*, page 17, 2013.

[HCBK12] Kai Huang, Gang Chen, C. Buckl, and A. Knoll. Conforming the runtime inputs for hard real-time embedded systems. In *2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 430 –436, June 2012.

[HDK+17] Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler, Falk Wurst, and Dirk Ziegenbein. Waters industrial challenge 2017. In *Proceedings of the 7th International Workshop on Analysis Tools and Methodologies for Embedded Real-Time Systems WATERS 2017*, 2017.

[HEQ+17] Zain A. H. Hammadeh, Rolf Ernst, Sophie Quinton, Rafik Henia, and Laurent Rioux. Bounding deadline misses in weakly-hard real-time systems with task dependencies. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 584–589, March 2017. ISSN: 1558-1101. doi:10.23919/DATE.2017.7927054.

[HHJ+05] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System Level Performance Analysis - the SymTA/S Approach. In *IEE Proceedings Computers and Digital Techniques*, 2005.

[HHK03] T.A. Henzinger, B. Horowitz, and C.M. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, January 2003. Conference Name: Proceedings of the IEEE. doi:10.1109/JPROC.2002.805825.

[HHM+16] J. Hennig, H. von Hasseln, H. Mohammad, S. Resmerita, S. Lukesch, and A. Naderlinger. Towards Parallelizing Legacy Embedded Control Software Using the LET Programming Paradigm. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 9–12, April 2016. doi:10.1109/RTAS.2016.7461355.

[HJRE06] Arne Hamann, Marek Jersak, Kai Richter, and Rolf Ernst. A framework for modular analysis and exploration of heterogeneous embedded systems. *Real-Time Systems*, 33(1-3):101–137, July 2006.

[HQE14] Z. A. H. Hammadeh, S. Quinton, and R. Ernst. Extending typical worst-case analysis using response-time dependencies to bound deadline misses. In *2014 International Conference on Embedded Software (EMSOFT)*, pages 1–10, October 2014. doi:10.1145/2656045.2656059.

[Infl12] Infineon. TriCore V1.6 - User Manual (Volume 1), May 2012. URL: https://www.infineon.com/dgdl/tc1_6__architecture_vol2.pdf?fileId=db3a3043372d5cc801374ad9c0653ad9.

[Int06] The International Electrotechnical Commission - IEC. *IEC 61025:2006 - Fault Tree Analysis (FTA)*, 2 edition, December 2006.

[Int11] International Organization for Standardization - ISO. *ISO/IEC 42010:2011 - Systems and software engineering – Architecture description*, Dec. 2011.

[Int18a] SAE International. J3016: Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems. https://www.sae.org/standards/content/j3016_201806/, 2018.

[Int18b] International Organization for Standardization - ISO. *ISO 26262:2018 - Road vehicles - Functional safety*, 2 edition, December 2018.

[Int19] International Standard Office. *ISO/PAS 21448:2019 Road vehicles : Safety of the intended functionality*. ISO, 2019.

[Int20] Intel. Product Brief Internet of Things - Intel Atom® Processor C3000 Series, November 2020. URL: https://www.intel.com/content/dam/www/public/us/en/documents/platform-briefs/atom-processor-c3000-series-embedded-iot-brief.pdf?asset=14757.

[JP86] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, January 1986. Publisher: Oxford Academic. URL: https://academic.oup.com/comjnl/article/29/5/390/486162, doi:10.1093/comjnl/29.5.390.

[Kin21] Colin Ian King. *stress-ng Version 0.11.07 (gcc 9.3, x86_64 Linux 5.6.19-rt12)*, 2021. URL: https://kernel.ubuntu.com/~cking/stress-ng/.

[Kos20] Adam Kostrzewa. Advanced Resource Management in Automotive Networks. In *DIVERSE: 2nd Workshop on Advanced Technologies in Vehicular Systems @ IEEE ETFA, Vienna Austria*, September 2020.

[KPWF19] Daniel Kästner, Markus Pister, Simon Wegener, and Christian Ferdinand. TimeWeaver: A Tool for Hybrid Worst-Case Execution Time Analysis. In Sebastian Altmeyer, editor, *19th International Workshop on Worst-Case Execution Time Analysis (WCET 2019)*, volume 72 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:11, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 2190-6807. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10766, doi:10.4230/OASIcs.WCET.2019.1.

[KQA+13] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, and F. J. Cazorla. Achieving timing composability with measurement-based probabilistic timing analysis. In *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, pages 1–8, June 2013. ISSN: 2375-5261. doi:10.1109/ISORC.2013.6913193.

[KRST11] Andreas Kern, Dominik Reinhard, Thilo Streichert, and Jürgen Teich. Gateway Strategies for Embedding of Automotive CAN-Frames into Ethernet-Packets and Vice Versa. In Mladen Berekovic, William Fornaciari, Uwe Brinkschulte, and Cristina Silvano, editors, *Architecture of Computing Systems - ARCS 2011*, number 6566 in Lecture Notes in Computer Science, pages 259–270. Springer Berlin Heidelberg, January 2011. URL: http://link.springer.com/chapter/10.1007/978-3-642-19137-4_22.

[KS12] Christoph M Kirsch and Ana Sokolova. The logical execution time paradigm. In *Advances in Real-Time Systems*. Springer, 2012.

[KSE20] Adam Kostrzewa, Dominik Stöhrmann, and Rolf Ernst. Towards Safety in Automotive Ethernet-based Networks with Dynamic Workloads. In *IEEE 6th World Forum on Internet of Things , New Orleans USA*, April 2020.

[Kuh] Dietrich Kuhlgatz. Safe Braking. URL: https://www.bosch.com/stories/beginnings-of-abs/ [cited 2021-03-13].

[Lee09] Edward A. Lee. Finite State Machines and Modal Models in Ptolemy II. Technical report, EECS Department, University of California, Berkeley, November 2009. URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-151.html.

[Lee17] Edward Ashford Lee. *Plato and the Nerd: The Creative Partnership of Humans and Technology*. The MIT Press, Cambridge, Massachusetts, October 2017. URL: https://mitpress.mit.edu/books/plato-and-nerd.

[Leh90] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 201–209, 1990. doi:10.1109/REAL.1990.128748.

[LGR+16] Mingsong Lv, Nan Guan, Jan Reineke, Reinhard Wilhelm, and Wang Yi. A Survey on Static Cache Analysis for Real-Time Systems. *Leibniz Transactions on Embedded Systems*, 3(1):05–1–05:48, June 2016. Number: 1. URL: https://ojs.dagstuhl.de/index.php/lites/article/view/LITES-v003-i001-a005, doi:10.4230/LITES-v003-i001-a005.

[LL73] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, January 1973. doi:10.1145/321738.321743.

[LS18] Edward A. Lee and Marjan Sirjani. What Good are Models? In Kyungmin Bae and Peter Csaba Ölveczky, editors, *Formal Aspects of Component Software*, Lecture Notes in Computer Science, pages 3–31, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-030-02146-7_1.

[LSD89] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *[1989] Proceedings. Real-Time Systems Symposium*, pages 166–171, December 1989. doi:10.1109/REAL.1989.63567.

[LTT] *LTTng – Linux Trace Toolkit Next Generation*, 2020-11 edition. URL: https://lttng.org/.

[Mat10] Johannes Matheis. *Abstraktionsebenenübergreifende Darstellung von Elektrik-Elektronik-Architekturen in Kraftfahrzeugen zur Ableitung von Sicherheitszielen nach ISO 26262*. Shaker, 2010.

[MBB18] Mario Maul, Gerhard Becker, and Ulrich Bernhard. Serviceorientierte EE-Zonenarchitektur Schlüsselelement für neue Marktsegmente. *ATZelektronik*, 13(1):36–41, February 2018. doi:10.1007/s35658-017-0105-3.

[MDA+19] Mischa Möstl, Alexander Dörflinger, Mark Albers, Harald Michalik, and Rolf Ernst. Self-Adaptation for Availability in CPU-FPGA Systems under Soft Errors. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2019.

[ME15] Mischa Möstl and Rolf Ernst. Cross-Layer Dependency Analysis for Safety-Critical Systems Design. In *ARCS 2015 - The 28th International Conference on Architecture of Computing Systems. Proceedings*, 2015. URL: https://ieeexplore.ieee.org/servlet/opac?punumber=7107091.

[ME16] Mischa Moestl and Rolf Ernst. Handling complex dependencies in system design. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1120–1123, 2016. URL: https://ieeexplore.ieee.org/document/7459476.

[ME17] Mischa Möstl and Rolf Ernst. Contracting challenges for system design and integration. *SIGBED Rev.*, 14(3):45–48, November 2017. doi:10.1145/3166227.3166236.

[ME18] Mischa Möstl and Rolf Ernst. Cross-Layer Dependency Analysis with Timing Dependence Graphs. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2018. doi:10.1109/DAC.2018.8465895.

[MG17] Luiz Eduardo G. Martins and Tony Gorschek. Requirements Engineering for Safety-Critical Systems: Overview and Challenges. *IEEE Software*, 34(4):49–57, 2017. Conference Name: IEEE Software. doi:10.1109/MS.2017.94.

[MHCE21] Mischa Möstl, Robin Hapka, Anika Christman, and Rolf Ernst. Timing diversity as a protective mechanism. *Under Submission to: ACM Transactions on Embedded Computing Systems (TECS)*, 2021. Submitted to EMSOFT'21.

[MHMJZ20] Martina Maggio, Arne Hamann, Eckart Mayer-John, and Dirk Ziegenbein. Control-System Stability Under Consecutive Deadline Misses Constraints. In Marcus Völp, editor, *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:24, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 1868-8969. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12384, doi:10.4230/LIPIcs.ECRTS.2020.21.

[MNSE19] Mischa Möstl, Marcus Nolte, Johannes Schlatow, and Rolf Ernst. Controlling Concurrent Change - A Multiview Approach Toward Updatable Vehicle Automation Systems. In Selma Saidi, Rolf Ernst, and Ed. Dirk Ziegenbein, editors, *Workshop on Autonomous Systems Design (ASD 2019)*, volume 68 of *OpenAccess Series in Informatics (OASIcs)*, pages 4:1–4:15, Florence, Italy, March 2019. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10337/.

[Mos58] Fred Moskowitz. The analysis of redundancy networks. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 77(5):627–632, November 1958. Conference Name: Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics. doi:10.1109/TCE.1958.6372698.

[MR] Inc. Mersenne Research. GIMPS - Free Prime95 software downloads - PrimeNet. URL: https://www.mersenne.org/download/ [cited 2020-11-25].

[MSB18] Jorge Martinez, Ignacio Sañudo, and Marko Bertogna. Analytical Characterization of End-to-End Communication Delays With Logical Execution Time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2244–2254, November 2018. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. doi:10.1109/TCAD.2018.2857398.

[MSE+16] Mischa Möstl, Johannes Schlatow, Rolf Ernst, Henry Hoffmann, Arif Merchant, and Alexander Shraer. Self-aware systems for the internet-of-things. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System*

Synthesis, CODES
16, Pittsburgh, Pennsylvania, October 2016. Special Session Paper. URL: https://doi.org/10.1145/2968456.2974043.

[MSE18a] Mischa Möstl, Johannes Schlatow, and Rolf Ernst. Synthesis of Monitors for Networked Systems With Heterogeneous Safety Requirements. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2824–2834, Nov 2018.

[MSE+18b] Mischa Möstl, Johannes Schlatow, Rolf Ernst, Nikil Dutt, Ahmed Nassar, Amir Rahmani, Fadi J. Kurdahi, Thomas Wild, Armin Sadighi, and Andreas Herkersdorf. Platform-Centric Self-Awareness as a Key Enabler for Controlling Changes in CPS. *Proceedings of the IEEE*, 106:1543–1567, September 2018. URL: https://doi.org/10.1109/JPROC.2018.2858023.

[MSN+16] Mischa Möstl, Johannes Schlatow, Marcus Nolte, Markus Maurer, and Rolf Ernst. Automating Future (Function-)Updates. In *Tagungsband ELIV-Marketplace: E/E im Pkw*, Düsseldorf, Germany, October 2016. VDI Wissensforum GmbH. ISBN: 978-3-945435-05-2.

[MSP+18] Mischa Möstl, Johannes Schlatow, Jonas Peeck, Rolf Ernst, Marcus Nolte, Inga Jatzkowski, Markus Maurer, and Anna Jankowski. Controlling Concurrent Change: Managing In-Field Change in Critical Embedded Systems. In *2018 12th European Workshop on Microelectronics Education (EWME)*, pages 107–112, September 2018. doi:10.1109/EWME.2018.8629451.

[MTE16] Mischa Möstl, Daniel Thiele, and Rolf Ernst. Towards Fail-operational Ethernet Based In-vehicle Networks. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16. ACM, 2016. URL: http://doi.acm.org/10.1145/2897937.2905021.

[NAME13] Moritz Neukirchner, Philip Axer, Tobias Michaels, and Rolf Ernst. Monitoring of Workload Arrival Functions for Mixed-Criticality Systems. pages 88–96. IEEE, December 2013.

[Nan12] Nancy G. Leveson. *Engineering a Safer World | Systems Thinking Applied to Safety*. The MIT Press, January 2012. Library Catalog: mitpress.mit.edu Publisher: The MIT Press. URL: https://mitpress.mit.edu/books/engineering-safer-world.

[Neu14] Moritz Neukirchner. *Establishing Sufficient Temporal Independence Efficiently: A Monitoring Approach*. Cuvillier Verlag, Göttingen, September 2014.

[NMA+12] M. Neukirchner, T. Michaels, P. Axer, S. Quinton, and R. Ernst. Monitoring Arbitrary Activation Patterns in Real-Time Systems. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 293–302, 2012. doi:10.1109/RTSS.2012.80.

[NNEB12] M. Neukirchner, M. Negrean, R. Ernst, and T. T. Bone. Response-time analysis of the flexray dynamic segment under consideration of slot-multiplexing. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, pages 21–30, June 2012. ISSN: 2150-3117. doi:10.1109/SIES.2012.6356566.

[NQEL13] Moritz Neukirchner, Sophie Quinton, Rolf Ernst, and Kai Lampka. Multi-mode monitoring for mixed-criticality real-time systems. pages 1–10. IEEE, September 2013.

[NSE09] Mircea Negrean, Simon Schliecker, and Rolf Ernst. Response-time analysis of arbitrarily activated tasks in multiprocessor systems with shared resources. In *Automation Test in Europe Conference Exhibition 2009 Design*, pages 524–529, April 2009. ISSN: 1558-1101. doi:10.1109/DATE.2009.5090720.

[OCOC20] Daniel Bristot de Oliveira, Daniel Casini, Rômulo Silva de Oliveira, and Tommaso Cucinotta. Demystifying the Real-Time Linux Scheduling Latency. In Marcus Völp, editor, *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:23, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 1868-8969. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12372, doi:10.4230/LIPIcs.ECRTS.2020.9.

[OO16] Daniel Bristot de Oliveira and Romulo Silva de Oliveira. Timing analysis of the PREEMPT RT Linux kernel. *Software: Practice and Experience*, 46(6):789–819, 2016. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2333. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2333, doi:10.1002/spe.2333.

[Par10] Gabriel Parmer. The Case for Thread Migration: Predictable IPC in a Customizable and Reliable OS. In *Proceedings 6th International*

Workshop on Operating Systems Platforms for Embedded Real-Time Applications, OSPERT 2010, Brussels, Belgium, 2010.

[PC10] DIRECTIVE 2001/95/EC OF THE EUROPEAN PARLIAMENT and OF THE COUNCIL. Directive 2001/95/ec of the european parliament and of the council of 3 december 2001 on general product safety, 2010. URL: https://eur-lex.europa.eu/eli/dir/2001/95/2010-01-01.

[PDB97] Sasikumar Punnekkat, Rob Davis, and Alan Burns. Sensitivity analysis of real-time task sets. In R. K. Shyamasundar and K. Ueda, editors, *Advances in Computing Science — ASIAN'97*, number 1345 in Lecture Notes in Computer Science, pages 72–82. Springer Berlin Heidelberg, January 1997. URL: http://link.springer.com/chapter/10.1007/3-540-63875-X_44.

[PS08] Peter Puschner and Martin Schoeberl. On Composable System Timing, Task Timing, and WCET Analysis. In Raimund Kirner, editor, *8th International Workshop on Worst-Case Execution Time Analysis (WCET'08)*, volume 8 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 2190-6807. URL: http://drops.dagstuhl.de/opus/volltexte/2008/1662, doi:10.4230/OASIcs.WCET.2008.1662.

[PSWT19] Behnaz Pourmohseni, Fedor Smirnov, Stefan Wildermann, and Jürgen Teich. Isolation-Aware Timing Analysis and Design Space Exploration for Predictable and Composable Many-Core Systems. In Sophie Quinton, editor, *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:24, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 1868-8969. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10749, doi:10.4230/LIPIcs.ECRTS.2019.12.

[QBH+14] Sophie Quinton, Torsten T. Bone, Julien Hennig, Moritz Neukirchner, Mircea Negrean, and Rolf Ernst. Typical Worst Case Response-Time Analysis and its Use in Automotive Network Design. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 1–6, San Francisco, CA, USA, June 2014. Association for Computing Machinery. doi:10.1145/2593069.2602977.

[Ram19] Eberle Rambo. *Fault-Tolerant Many-Cores for Mixed-Critical Real-Time Systems.* PhD thesis, Technische Universität Braunschweig, March 2019. ISBN: 9781067701789. URL: https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00066436?q=rambo, doi:10.24355/dbbs.084-201903041233-0.

[RE15] E. A. Rambo and R. Ernst. Worst-case communication time analysis of networks-on-chip with shared virtual channels. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 537–542, March 2015. ISSN: 1558-1101. doi:10.7873/DATE.2015.0023.

[RE17] Eberle A. Rambo and Rolf Ernst. Replica-Aware Co-Scheduling for Mixed-Criticality. In Marko Bertogna, editor, *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, volume 76 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:20, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 1868-8969. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7152, doi:10.4230/LIPIcs.ECRTS.2017.20.

[Ren] Renesas. R-Car H3 System-on-Chip (SoC). https://www.renesas.com/kr/en/solutions/automotive/soc/r-car-h3.html. URL: https://www.renesas.com/kr/en/solutions/automotive/soc/r-car-h3.html.

[Ric05] Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models.* Institut für Datentechnik, 2005. URL: http://www.digibib.tu-bs.de/?docid=00001765.

[RJE05] R. Racu, M. Jersak, and R. Ernst. Applying sensitivity analysis in real-time distributed systems. In *11th IEEE Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005*, pages 160–169, March 2005. doi:10.1109/RTAS.2005.10.

[RMF19] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. The Real-Time Linux Kernel: A Survey on PREEMPT_rt. *ACM Computing Surveys*, 52(1):18:1–18:36, February 2019. doi:10.1145/3297714.

[SBM16] T. Stolte, G. Bagschik, and M. Maurer. Safety goals and functional safety requirements for actuation systems of automated vehicles. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2191–2198, November 2016.

[SCB+13] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Zirkler, L. Fiege, M. Armbruster, G. Spiegelberg, and A. Knoll. RACE: A Centralized Platform Computer Based Architecture for Automotive Applications. In *Electric Vehicle Conference (IEVC), 2013 IEEE International*, pages 1–6, October 2013. doi:10.1109/IEVC.2013.6681152.

[SCH] *sched(7) — Linux manual page*, 2019-08-02 edition.

[Sch11] Simon Schliecker. *Performance Analysis of Multiprocessor Real-Time Systems with Shared Resources*. Cuvillier, Göttingen, 1 edition, April 2011.

[SE09a] Simon Schliecker and Rolf Ernst. A recursive approach to end-to-end path latency computation in heterogeneous multiprocessor systems. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '09, pages 433–442, New York, NY, USA, October 2009. Association for Computing Machinery. doi:10.1145/1629435.1629494.

[SE09b] M. Sebastian and R. Ernst. Reliability Analysis of Single Bus Communication with Real-Time Requirements. In *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 3–10, November 2009. doi:10.1109/PRDC.2009.10.

[SE16] Johannes Schlatow and Rolf Ernst. Response-time analysis for task chains in communicating threads. In *Real-Time Embedded Technology & Applications Symposium (RTAS)*, Vienna, Austria, April 2016.

[SE17] Johannes Schlatow and Rolf Ernst. Response-time analysis for task chains with complex precedence and blocking relations. *International Conference on Embedded Software (EMSOFT), ACM Transactions on Embedded Computing Systems ESWEEK Special Issue*, 16(5s):172:1–172:19, sep 2017.

[Sif18] Joseph Sifakis. System design in the era of iot - meeting the autonomy challenge. In Simon Bliudze and Saddek Bensalem, editors, *Proceedings of the 1st International Workshop on Methods and Tools for Rigorous System Design, MeTRiD@ETAPS 2018, Thessaloniki, Greece, 15th April 2018*, volume 272 of *EPTCS*, pages 1–22, 2018. doi:10.4204/EPTCS.272.1.

[Sim] Bettina Simon. A backbone of Automotive Electronics. URL: https://www.bosch.com/stories/

50-years-of-bosch-gasoline-injection-jetronic/ [cited 2021-03-13].

[SK07] Raja Sengupta and Christoph Kirsch. The Evolution of Real-Time Programming. In Sang Son, Insup Lee, and Joseph Y-T. Leung, editors, *Handbook of Real-Time and Embedded Systems*, volume 20073969, pages 11–1–11–23. Chapman and Hall/CRC, July 2007. Series Title: Chapman & Hall/CRC Computer & Information Science Series. URL: http://www.crcnetbase.com/doi/abs/10.1201/9781420011746.ch11, doi:10.1201/9781420011746.ch11.

[SME15] Johannes Schlatow, Mischa M"ostl, and Rolf Ernst. An extensible autonomous reconfiguration framework for complex component-based embedded systems. In *12th International Conference on Autonomic Computing (ICAC)*, pages 239–242, Grenoble, France, July 2015.

[SME+17] Johannes Schlatow, Mischa Möstl, Rolf Ernst, Marcus Nolte, Inga Jatzkowski, Markus Maurer, Christian Herber, and Andreas Herkersdorf. Self-awareness in autonomous automotive systems. In *Design, Automation and Test in Europe (DATE)*, Lausanne, Switzerland, March 2017. URL: https://doi.org/10.24355/dbbs.084-201803221524.

[SME19] Johannes Schlatow, Mischa Möstl, and Rolf Ernst. Self-aware scheduling for mixed-criticality component-based systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Montreal, Canada, April 2019.

[SMT+18] Johannes Schlatow, Mischa Möstl, Sebastian Tobuschat, Tasuku Ishigooka, and Rolf Ernst. Data-age analysis and optimisation for cause-effect chains in automotive control systems. In *IEEE Symposium on Industrial Embedded Systems (SIES)*, Graz, Austria, June 2018.

[SNE09] Simon Schliecker, Mircea Negrean, and Rolf Ernst. Response Time Analysis on Multicore ECUs With Shared Resources. *IEEE Transactions on Industrial Informatics*, 5(4):402–413, November 2009. Conference Name: IEEE Transactions on Industrial Informatics. doi:10.1109/TII.2009.2032068.

[SNM+17] Johannes Schlatow, Marcus Nolte, Mischa Möstl, Inga Jatzkowski, Rolf Ernst, and Markus Maurer. Towards model-based integration of component-based automotive software systems. In *Annual*

Conference of the IEEE Industrial Electronics Society (IECON17), Beijing, China, October 2017. URL: https://doi.org/10.24355/dbbs.084-201803221525.

[SP15] J. Song and G. Parmer. C'Mon: a predictable monitoring infrastructure for system-level latent fault detection and recovery. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 247–258, April 2015. doi:10.1109/RTAS.2015.7108448.

[SRIE08] Simon Schliecker, Jonas Rox, Matthias Ivers, and Rolf Ernst. Providing accurate event models for the analysis of heterogeneous multiprocessor systems. In *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, CODES+ISSS '08, pages 185–190, New York, NY, USA, 2008. ACM.

[SSE05] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 41–48, July 2005. ISSN: 2377-5998. doi:10.1109/ECRTS.2005.26.

[SVZ15] Bernhard Schätz, Sebastian Voss, and Sergey Zverlov. Automating design-space exploration: optimal deployment of automotive SW-components in an ISO26262 context. In *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, pages 1–6, New York, NY, USA, June 2015. Association for Computing Machinery. doi:10.1145/2744769.2747912.

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*, pages 661–666. Addison-Wesley Professional, 4th edition, 2011.

[SWP13] J. Song, J. Wittrock, and G. Parmer. Predictable, Efficient System-Level Fault Tolerance in C^3. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 21–32, December 2013. doi:10.1109/RTSS.2013.11.

[TAE15] Daniel Thiele, Philip Axer, and Rolf Ernst. Improving Formal Timing Analysis of Switched Ethernet by Exploiting FIFO Scheduling. In *52nd Annual Design Automation Conference*, DAC '15, pages 41:1–41:6, New York, NY, USA, 2015. ACM.

[TAED13] Sebastian Tobuschat, Philip Axer, Rolf Ernst, and Jonas Diemer. *IDAMC: A NoC for mixed criticality systems*. IEEE Computer Society, August 2013. ISSN: 1533-2306 Pages: 149-156. URL: https://www.computer.org/csdl/proceedings-article/rtcsa/2013/06732214/12OmNzuIjtp, doi:10.1109/RTCSA.2013.6732214.

[TBW94] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994. URL: http://www.springerlink.com/content/j4684m3021l003t6/abstract/, doi:10.1007/BF01088593.

[TCN00] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *The IEEE Intern. Symposium on Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva*, 2000.

[TE16a] D. Thiele and R. Ernst. Formal worst-case performance analysis of time-sensitive Ethernet with frame preemption. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–9, September 2016. doi:10.1109/ETFA.2016.7733740.

[TE16b] D. Thiele and R. Ernst. Formal worst-case timing analysis of Ethernet TSN's burst-limiting shaper. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 187–192, March 2016. ISSN: 1558-1101.

[Tem07] Josef Templ. Timing definition language (TDL) 1.5 specification. Technical report, University of Salzburg, 2007. URL: https://www.softwareresearch.net/fileadmin/src/docs/publications/T024.pdf [cited 2021-02-15].

[The10] The International Electrotechnical Commission - IEC. *IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2 edition, April 2010.

[The18] The International Electrotechnical Commission - IEC. *IEC 60812:2018 - Failure modes and effects analysis (FMEA and FMECA)*, 3 edition, August 2018.

[TJJG19] Y. Tang, Y. Jiang, X. Jiang, and N. Guan. Pay-Burst-Only-Once in Real-Time Calculus. In *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*,

pages 1–6, August 2019. ISSN: 2325-1301. doi:10.1109/RTCSA.2019.8864582.

[TSAE16a] Daniel Thiele, Johannes Schlatow, Philip Axer, and Rolf Ernst. Formal Timing Analysis of CAN-to-Ethernet Gateway Strategies in Automotive Networks. *Real-Time Syst.*, 52(1):88–112, January 2016. URL: http://dx.doi.org/10.1007/s11241-015-9243-y, doi:10.1007/s11241-015-9243-y.

[TSAE16b] Daniel Thiele, Johannes Schlatow, Philip Axer, and Rolf Ernst. Formal timing analysis of CAN-to-Ethernet gateway strategies in automotive networks. *Real-Time Systems*, 52(1):88–112, January 2016. doi:10.1007/s11241-015-9243-y.

[TSAE16c] Daniel Thiele, Johannes Schlatow, Philip Axer, and Rolf Ernst. Formal timing analysis of can-to-ethernet gateway strategies in automotive networks. *Real-Time Syst.*, 52(1):88–112, January 2016. URL: http://dx.doi.org/10.1007/s11241-015-9243-y.

[Vec] Vector Informatik. PREEvision | Die E/E-Engineering-Lösung. Library Catalog: www.vector.com. URL: https://www.vector.com/de/de/produkte/produkte-a-z/software/preevision/ [cited 2020-08-04].

[VEH14] Sebastian Voss, Johannes Eder, and Florian Holzl. Design Space Exploration and its Visualization in AUTOFOCUS3. page 10, 2014.

[Ves07] S. Vestal. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243, December 2007. doi:10.1109/RTSS.2007.47.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):36:1–36:53, May 2008. doi:10.1145/1347375.1347389.

[WGR+09] Reinhard Wilhelm, Daniel Grund, Jan Reineke, Marc Schlickling, Markus Pister, and Christian Ferdinand. Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-Critical

Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):966–978, July 2009. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. doi:10.1109/TCAD.2009.2013287.

[WHLS16] Hermann Winner, Hakuli, Stephan, Lotz, Felix, and Singer, Christina, editors. *Handbook of Driver Assistance Systems*. Springer International Publishing, 2016. URL: https://www.springer.com/de/book/9783319123516.

[Wil18] Reinhard Wilhelm. Mixed Feelings About Mixed Criticality (Invited Paper). In Florian Brandner, editor, *18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018)*, volume 63 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:9, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 2190-6807. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9747, doi:10.4230/OASIcs.WCET.2018.1.

[WKP13] Z. P. Wu, Y. Krish, and R. Pellizzoni. Worst Case Analysis of DRAM Latency in Multi-requestor Systems. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 372–383, December 2013. ISSN: 1052-8725. doi:10.1109/RTSS.2013.44.

[WL96] D. E. Wrege and J. Liebherr. Video traffic characterization for multimedia networks with a deterministic service. In *Proceedings IEEE INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation*, volume 2, pages 537–544 vol.2, March 1996. doi:10.1109/INFCOM.1996.493346.

[WLB+18] Timo Woopen, Bastian Lampe, Torben Böddeker, Lutz Eckstein, Alexandru Kampmann, Bassam Alrifaee, Stefan Kowalewski, Dieter Moormann, Torben Stolte, Inga Jatzkowski, Markus Maurer, Mischa Möstl, Rolf Ernst, Stefan Ackermann, Christian Amersbach, Hermann Winner, Dominik Püllen, Stefan Katzenbeisser, Stefan Leinen, Matthias Becker, Christoph Stiller, Kai Furmans, Klaus Bengler, Frank Diermeyer, Markus Lienkamp, Dan Keilhoff, Hans-Christian Reuss, Michael Buchholz, Klaus Dietmayer, Henning Lategahn, Norbert Siepenkötter, Martin Elbs, Edgar v. Hinüber, Marius Dupuis, and Christian Hecker. *UNICARagil - Disruptive Modular Architectures for Agile, Automated Vehicle Concepts*. Institute for Automotive Engineering, RWTH Aachen ; Aachen : Institute for Combustion Engines, RWTH Aachen, Aachen, 1.

edition edition, 2018. Medium: online, print Meeting Name: 27th Aachen Colloquium Automobile and Engine Technology 2018.

[WTVL06] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. System architecture evaluation using modular performance analysis: a case study. *Intern. Journal on Software Tools for Technology Transfer*, 8(6):649–667, July 2006.

[XHK+15] Wenbo Xu, Zain A.H. Hammadeh, Alexander Kröller, Rolf Ernst, and Sophie Quinton. Improved Deadline Miss Models for Real-Time Systems Using Typical Worst-Case Analysis. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 247–256, July 2015. ISSN: 2377-5998. doi:10.1109/ECRTS.2015.29.

[Yos20] Junko Yoshida. EETimes - Unveiled: BMW's Scalable AV Architecture -, April 2020. URL: https://www.eetimes.com/unveiled-bmws-scalable-av-architecture/ [cited 2020-12-19].

List of Figures

List of Tables

List of Algorithms

List of Acronyms

ACC	Adaptive Cruise Control
ADAS	Advanced Driver Assistance Systems
ADL	architecture description language
AFDX	Avionics Full-Duplex Ethernet
ASIL	Automotive Safety Integrity Level
BCET	best-case execution time
BCRT	best-case response time
BET	bounded execution time
BFS	breadth-first search
CAN	Controller Area Network
CCC	Controlling Concurrent Change
CCF	common cause failure
CET	core execution time
CLM	cross-layer model
COTS	commercial off-the-shelf
CPA	compositional performance analysis
CPS	Cyber-Physical System
DAL	Design Assurance Level
DDS	data distribution service
DFG	Deutsche Forschungsgemeinschaft
DFS	depth-first search
DMR	dual modular redundancy
DRAM	dynamic random access memory
DSE	design space exploration
ECU	electronic control unit
EDA	electronic design automation
FMEA	Failure Mode and Effects Analysis
FTA	Fault Tree Analysis

FTTI fault-tolerant time interval
HARA hazard analysis and risk assessment
IMU inertial measurement unit
IoT Internet of Things
LET logical execution time
LKA lane keeping assistant
MCC Multi-Change Controller
MC mixed-criticality
MMU memory management unit
MPA modular performance analysis
MPSoC multi-processor system-on-chip
MTTF mean-time to failure
NoC network-on-chip
OEM Original Equipement Manufacturer
OS operating system
PFH probability of failure per hour
QM quality managed
RBD reliability block diagram
RM rate monotonic
ROS robot operating system
RTE Run Time Environment
SEooC safety element out of context
SIL Safety Integrity Level
SL-LET system-level logical execution time
SMT simultaneous multi-threading
SOTIF Safety of the Intended Functionality
SPNP static-priority non-preemptive
SPP static-priority preemptive
SoC system-on-chip
TDG timing-dependence graph
TDM time-division multiplex
TLB translation lookaside buffer
TMR Triple Modular Redundancy
TWCA typical worst-case analysis
VM virtual machine
WCET worst-case execution time
WCRT worst-case response time
pWCET probabilistic worst-case execution time

List of Own Publications

This appendix lists the publications of the author of this thesis. Publications are grouped into those with relation to the thesis and those without.

Publications with relation to the thesis

Under Submission

Reference [MHCE21]:

Mischa Möstl, Robin Hapka, Anika Christman, and Rolf Ernst. Timing diversity as a protective mechanism. *Under Submission to: ACM Transactions on Embedded Computing Systems (TECS)*, 2021. Submitted to EMSOFT'21

In this paper the idea of timing diversity, presented in Chapter 5, to increase timing reliability is to be published. The paper investigates the statistical independence assumption of timing diversity, based on statistical tests of a simple benchmark application and the case study from Section 5.4. This paper is under submission to the EMSOFT'21 conference, which is part of Embedded Systems Week (ESWEEK)[1].

Reviewed

Reference [ME18]:

Mischa Möstl and Rolf Ernst. Cross-Layer Dependency Analysis with Timing Dependence Graphs. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2018. doi:10.1109/DAC.2018.8465895

[1] esweek.org

The paper presents the idea of the TDGs for timing dependency analysis. Its is one of the pillars of Chapter 2.

Reference [MSE18a]:

Mischa Möstl, Johannes Schlatow, and Rolf Ernst. Synthesis of Monitors for Networked Systems With Heterogeneous Safety Requirements. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2824–2834, Nov 2018

This work introduced the idea of synthesizing a network of monitors system-wide to ensure timing safety of the entire system. It is the foundation of the steps in Chapter 3.

Reference [MNSE19]:

Mischa Möstl, Marcus Nolte, Johannes Schlatow, and Rolf Ernst. Controlling Concurrent Change - A Multiview Approach Toward Updatable Vehicle Automation Systems. In Selma Saidi, Rolf Ernst, and Ed. Dirk Ziegenbein, editors, *Workshop on Autonomous Systems Design (ASD 2019)*, volume 68 of *OpenAccess Series in Informatics (OASIcs)*, pages 4:1–4:15, Florence, Italy, March 2019. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10337/

The paper presents ideas and initial solutions for a coherent safety guided integration flow for safety requirements from behavioral requirements, down to technical safety requirements. The derived technical requirements are verified using the analysis flow from Section 2.1 and necessary design adaptions synthesized through the mechanisms from Chapter 3.

Reference [SME19]: Johannes Schlatow, Mischa Möstl, and Rolf Ernst. Self-aware scheduling for mixed-criticality component-based systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Montreal, Canada, April 2019

This paper is another result of the CCC project. The idea developed in this paper follows a different approach than the one taken in this thesis for dealing with uncertainty of model parameters. It introduces an at run-time feedback mechanism to tune model parameters to actual observed behavior. By in-system analysis of the resulting model, scheduling parameters are adapted to maintain safe operation of the system. It is a complementary strategy to the one described in Chapter 3.

Reference [SMT+18]:

Johannes Schlatow, Mischa Möstl, Sebastian Tobuschat, Tasuku Ishigooka, and Rolf Ernst. Data-age analysis and optimisation for cause-effect chains

in automotive control systems. In *IEEE Symposium on Industrial Embedded Systems (SIES)*, Graz, Austria, June 2018

Based on a cooperation with industry, this paper shows that data-age analysis alone is not sufficient for a predictable design. It influenced the thoughts laid down in Chapter 4.

Reference [BME16]:

Matthias Beckert, Mischa Möstl, and Rolf Ernst. Zero-time communication for automotive multi-core systems under SPP scheduling. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–9, September 2016. doi:10.1109/ETFA.2016.7733563

Implementation of resource local LET through a double buffering strategy. It is one of many possibilities to implement resource local LET but its principle can be extended to SL-LET as described in Section 4.5.

Reference [ME16]:

Mischa Moestl and Rolf Ernst. Handling complex dependencies in system design. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1120–1123, 2016. URL: https://ieeexplore.ieee.org/document/7459476

This publication as well as [ME15] presents the idea of cross-layer dependency analysis as presented in Section 2.1.

Reference [ME15]:

Mischa Möstl and Rolf Ernst. Cross-Layer Dependency Analysis for Safety-Critical Systems Design. In *ARCS 2015 - The 28th International Conference on Architecture of Computing Systems. Proceedings*, 2015. URL: https://ieeexplore.ieee.org/servlet/opac?punumber=7107091

This publication as well as [ME16] presents the idea of cross-layer dependency analysis as presented in Section 2.1.

Publications without relation to the thesis

Reviewed

Reference [MDA+19]:

Mischa Möstl, Alexander Dörflinger, Mark Albers, Harald Michalik, and Rolf Ernst. Self-Adaptation for Availability in CPU-FPGA Systems under Soft Errors. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2019

The publication presents results from the CCC project. It utilizes the cross-layer model introduced in Section 2.1 for a reliability estimation of a CPU-FPGA system under soft-errors. The analysis is designed to be an analysis engine for the MCC mechanism for concurrent change (cf. Section 1.2.4).

Reference [DAF+19]:

Alexander Dörflinger, Mark Albers, Björn Fiethe, Harald Michalik, Mischa Möstl, Johannes Schlatow, and Rolf Ernst. Demonstrating Controlled Change for Autonomous Space Vehicles. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2019

In this paper one out of the three demonstrator of the CCC project is described. This demonstration e.g. contains the mechanisms from described in [MDA+19] and [SME19].

Reference [MSE+18b]:

Mischa Möstl, Johannes Schlatow, Rolf Ernst, Nikil Dutt, Ahmed Nassar, Amir Rahmani, Fadi J. Kurdahi, Thomas Wild, Armin Sadighi, and Andreas Herkersdorf. Platform-Centric Self-Awareness as a Key Enabler for Controlling Changes in CPS. *Proceedings of the IEEE*, 106:1543–1567, September 2018. URL: https://doi.org/10.1109/JPROC.2018.2858023

This paper describes self-awareness aspects of the automatic integration flow from the CCC project in context with other research. It presents why the mechanisms developed there are key enablers for future self-aware systems.

Reference [SME+17]:

Johannes Schlatow, Mischa Möstl, Rolf Ernst, Marcus Nolte, Inga Jatzkowski, Markus Maurer, Christian Herber, and Andreas Herkersdorf. Self-awareness in autonomous automotive systems. In *Design, Automation and Test in Europe (DATE)*, Lausanne, Switzerland, March 2017. URL: https://doi.org/10.24355/dbbs.084-201803221524

This paper focuses the scope of [MSE+16] to the needs and challenges for automotive systems. The challenges and gains of self-awareness for integration was discussed in more detail in [MSE+18b].

Reference [SNM+17]: Johannes Schlatow, Marcus Nolte, Mischa Möstl, Inga Jatzkowski, Rolf Ernst, and Markus Maurer. Towards model-based integration of component-based automotive software systems. In *Annual Conference of the IEEE Industrial Electronics Society (IECON17)*, Beijing, China, October 2017. URL: https://doi.org/10.24355/dbbs.084-201803221525

Presents aspects of the model-based integration flow in the CCCs project on the example of CCC's automotive showcase. It outlines the idea of an analysis engine that particularly checks safety constraints based on a cross-layer model of the system.

Reference [GSME17]:

Kai-Björn Gemlau, Johannes Schlatow, Mischa Möstl, and Rolf Ernst. Compositional analysis for the waters industrial challenge 2017. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Dubrovnik, Croatia, June 2017

Presents the analysis of a combustion engine case study system – among other programming paradigms, also plain LET for a multi-core system was investigated. Lead to the thoughts that culminated in Chapter 4.

Reference [MSN+16]:

Mischa Möstl, Johannes Schlatow, Marcus Nolte, Markus Maurer, and Rolf Ernst. Automating Future (Function-)Updates. In *Tagungsband ELIV-Marketplace: E/E im Pkw*, Düsseldorf, Germany, October 2016. VDI Wissensforum GmbH. ISBN: 978-3-945435-05-2

This publication presents an outlook of how the CCC mechanisms and particularly the integration flow can be applied in the context of automotive platforms and systems.

Reference [MTE16]:

Mischa Möstl, Daniel Thiele, and Rolf Ernst. Towards Fail-operational Ethernet Based In-vehicle Networks. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16. ACM, 2016. URL: http://doi.acm.org/10.1145/2897937.2905021

Based on cooperation with industry, this paper highlights results and conclusions drawn from analyzing in-vehicle Ethernet setups.

Reference [MSE+16]:

Mischa Möstl, Johannes Schlatow, Rolf Ernst, Henry Hoffmann, Arif Merchant, and Alexander Shraer. Self-aware systems for the internet-of-things. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES
́16, Pittsburgh, Pennsylvania, October 2016. Special Session Paper. URL: https://doi.org/10.1145/2968456.2974043

This paper presents the CCC approach in the context of self-aware systems. The initial ideas of this paper where later extended in [MSE+18b].

Reference [ME17]:

Mischa Möstl and Rolf Ernst. Contracting challenges for system design and integration. *SIGBED Rev.*, 14(3):45–48, November 2017. doi:10.1145/3166227.3166236

This publication is a result of CCC and describes challenges of the applicability of contracting (based on assumptions and guarantees) for cyber-physical systems design.

Reference [SME15]: Johannes Schlatow, Mischa M"ostl, and Rolf Ernst. An extensible autonomous reconfiguration framework for complex component-based embedded systems. In *12th International Conference on Autonomic Computing (ICAC)*, pages 239–242, Grenoble, France, July 2015

The paper presents the early design of the MCC architecture from project CCC. My contribution is the aspect of safety in the presented analysis.

Reference [MSP+18]:

Mischa Möstl, Johannes Schlatow, Jonas Peeck, Rolf Ernst, Marcus Nolte, Inga Jatzkowski, Markus Maurer, and Anna Jankowski. Controlling Concurrent Change: Managing In-Field Change in Critical Embedded Systems. In *2018 12th European Workshop on Microelectronics Education (EWME)*, pages 107–112, September 2018. doi:10.1109/EWME.2018.8629451

This publication summarizes some contributions of the CCC project. It focuses on the impact of CCC's findings on teaching.

Unreviewed

Reference [WLB+18]: Timo Woopen, Bastian Lampe, Torben Böddeker, Lutz Eckstein, Alexandru Kampmann, Bassam Alrifaee, Stefan Kowalewski, Dieter Moormann, Torben Stolte, Inga Jatzkowski, Markus Maurer, Mischa

Möstl, Rolf Ernst, Stefan Ackermann, Christian Amersbach, Hermann Winner, Dominik Püllen, Stefan Katzenbeisser, Stefan Leinen, Matthias Becker, Christoph Stiller, Kai Furmans, Klaus Bengler, Frank Diermeyer, Markus Lienkamp, Dan Keilhoff, Hans-Christian Reuss, Michael Buchholz, Klaus Dietmayer, Henning Lategahn, Norbert Siepenkötter, Martin Elbs, Edgar v. Hinüber, Marius Dupuis, and Christian Hecker. *UNICARagil - Disruptive Modular Architectures for Agile, Automated Vehicle Concepts*. Institute for Automotive Engineering, RWTH Aachen ; Aachen : Institute for Combustion Engines, RWTH Aachen, Aachen, 1. edition edition, 2018. Medium: online, print Meeting Name: 27th Aachen Colloquium Automobile and Engine Technology 2018

The paper describes the envisioned demo vehicles for the UNICARagil[1] *project as well as further project goals and challenges for highly automated driving. My contribution are aspects to the safety of the hardware and software platform.*

[1] unicaragil.de